Costas Iliopoulos
Hon Wai Leong
Wing-Kin Sung (Eds.)

LNCS 10979

Combinatorial Algorithms

29th International Workshop, IWOCA 2018
Singapore, July 16–19, 2018
Proceedings

Springer

Lecture Notes in Computer Science 10979

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

More information about this series at http://www.springer.com/series/7407

Costas Iliopoulos · Hon Wai Leong
Wing-Kin Sung (Eds.)

Combinatorial Algorithms

29th International Workshop, IWOCA 2018
Singapore, July 16–19, 2018
Proceedings

Springer

Editors
Costas Iliopoulos
Department of Informatics
King's College London
London
UK

Hon Wai Leong
Department of Computer Science,
School of Computing
National University of Singapore
Singapore
Singapore

Wing-Kin Sung
Department of Computer Science,
School of Computing
National University of Singapore
Singapore
Singapore

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-94666-5 ISBN 978-3-319-94667-2 (eBook)
https://doi.org/10.1007/978-3-319-94667-2

Library of Congress Control Number: 2018939453

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 proceedings volume contains papers presented at IWOCA 2018, the 29th International Workshop on Combinatorial Algorithms, held during July 16–19, 2018, at the Department of Computer Science, National University of Singapore (NUS), Singapore. The conference covered diverse areas of combinatorial algorithms, namely, complexity theory, graph theory and combinatorics, combinatorial optimization, cryptography and information security, algorithms on strings and graphs, graph drawing and labelling, computational algebra and geometry, computational biology, probabilistic and randomized algorithms, algorithms for big data analytics, and new paradigms of computation. The conference was organized by the School of Computing, National University of Singapore.

IWOCA is an annual conference series on all aspects of combinatorial algorithms. The series of IWOCA conferences grew out of over 28 years of history. Initially, the conference was a local workshop in Australia, known as AWOCA. In 2007, it became an international conference. Previous meetings have been held in Australia, Canada, Czech Republic, Finland, France, Indonesia, India, Italy, Japan, South Korea, UK, and USA. IWOCA is led by a strong Steering Committee, whose members are Charles Colbourn (Arizona State University), Costas Iliopoulos (King's College), and Bill Smyth (McMaster University). The Program Committees comprise computer scientists of international repute from different parts of the globe. Notably, the Program Committee of IWOCA 2018 comprised 46 eminent researchers from Australia, Bangladesh, Canada, Chile, China, Czech Republic, Finland, France, Greece, Hong Kong, Israel, Italy, Japan, Malaysia, Norway, Singapore, Slovenia, Taiwan, UK, and USA.

The technical program was finalized by selecting the highest-quality papers from among 69 submitted papers. After a rigorous review followed by in-depth discussion by the Program Committee, this year we could only accept 31 high-quality papers. Among these 31 papers, we selected "Linear Ramsey Numbers" for the best paper award, which presented at the conference. In addition to the 31 contributed talks, the scientific program of the workshop included invited talks by three eminent researchers, namely, Prof. Michael Fellows (University of Bergen, Norway), Prof. Sanjay Jain (National University of Singapore, Singapore), and Prof. Kunihiko Sadakane (University of Tokyo, Japan). We are extremely grateful to our invited speakers for their excellent talks at the workshop. We thank all the authors who submitted their works for consideration to IWOCA 2018. We deeply appreciate the contribution of all Program Committee members and external reviewers for handling the submissions in a timely manner despite their extremely busy schedule. We would like to acknowledge the EasyChair conference management system again for providing us with their celebrated platform for conference administration. We are grateful to Springer for

publishing the proceedings of IWOCA 2018 in the LNCS series. As always, we are deeply indebted to the IWOCA Steering Committee for their continuous guidance, support, and leadership. Above all, we are extremely grateful to the Organizing Committee of IWOCA 2018 for making the event a grand success. Finally, we would like to thanks Springer for sponsoring the conference.

July 2018

Costas Iliopoulos
Hon Wai Leong
Wing-Kin Sung

Organization

Steering Committee

Charles Colbourn	Arizona State University, USA
Costas Iliopoulos	King's College, UK
Bill Smyth	McMaster University, Canada

Program Committee

Donald Adjeroh	West Viriginia University, USA
Cristina Bazgan	Paris Dauphine University, France
Yeow Meng Chee	Nanyang Technological University, Singapore
Kwok Pui Choi	National University of Singapore, Singapore
Charles Colbourn	Arizona State University, USA
Bhaskar Dasgupta	University of Illinois at Chicago, USA
Vlad Estivill-Castro	Griffith University, Australia
Gabriele Fici	Università di Palermo, Italy
Dalibor Froncek	University of Minnesota, USA
Travis Gagie	Diego Portales University, Chile
Serge Gaspers	UNSW Sydney and Data61, CSIRO, Australia
Pinar Heggernes	University of Bergen, Norway
Wing-Kai Hon	National Tsing Hua University, Taiwan
Seok-Hee Hong	University of Sydney, Australia
Peter Horak	University of Washington, USA
Costas Iliopoulos	King's College, UK
Jesper Jansson	The Hong Kong Polytechnic University, SAR China
Ralf Klasing	CNRS and University of Bordeaux, France
Jan Kratochvil	Charles University, Czech Republic
Dieter Kratsch	University of Lorraine, France
Thierry Lecroq	University of Rouen, France
Hon Wai Leong	National University of Singapore, Singapore
Zsuzsanna Liptak	University of Verona, Italy
Martin Milanič	University of Primorska, Slovenia
Lucia Moura	University of Ottawa, Canada
Veli Mäkinen	University of Helsinki, Finland
Gonzalo Navarro	University of Chile, Chile
Yen Kaow Ng	Tunku Abdul Rahman University, Malaysia
Hirotaka Ono	Nagoya University, Japan
Patric Ostergard	Aalto University, Finland
Vangelis Paschos	Paris Dauphine University, France
Solon Pissis	King's College, UK
Simon Puglisi	University of Helsinki, Finland

M. Sohel Rahman	Bangladesh University of Engineering and Technology, Bangladesh
Hiroshi Sakamoto	Kyushu Institute of Technology, Japan
Jamie Simpson	Curtin University, Australia
Michiel Smid	Carleton University, Canada
Sagi Snir	University of Haifa, Israel
Wing-Kin Sung	National University of Singapore, Singapore
Ryuhei Uehara	Japan Advanced Institute of Science and Technology, Japan
Ugo Vaccaro	University of Salerno, Italy
Lusheng Wang	City University of Hong Kong, SAR China
Sue Whitesides	University of Victoria, Canada
Mingyu Xiao	University of Science and Technology of China, China
Christos Zaroliagis	University of Patras, Greece
Louxin Zhang	National University of Singapore, Singapore

Additional Reviewers

Abdeddaim, Said
Arroyuelo, Diego
Arumugam, Subramanian
Aziz, Haris
Belov, Alexander
Charalampopoulos, Panagiotis
Cicalese, Ferdinando
Cichacz, Sylwia
Cordasco, Gennaro
Courcelle, Bruno
Dasler, Philip
Della Croce, Federico
Demange, Marc
Dujmovic, Vida
García Quiles, Sergio
Gawrychowski, Pawel
Giannakos, Aristotelis
Golovach, Petr
Hama, Vitor
Haslegrave, John
Holub, Stepan
Hu, Yannan
Huang, Shenwei
Huber, Katharina
Hung, Ling-Ju
Kempa, Dominik
Khramtcova, Elena
Kola, Srinivasa Rao
Kupferman, Orna
Kwon, O-Joung
Lampis, Michael
Lee, Chia-Wei
Lin, Weibo
Maenhaut, Barbara
Mampentzidis, Konstantinos
Manea, Florin
Manlove, David
Manzini, Giovanni
Martínez, Luis
McKay, Brendan
Miyazaki, Shuichi
Musco, Christopher
Najeebullah, Kamran
Ochem, Pascal
Ochoa, Carlos
Papadopoulos, Charis
Peters, Dominik
Radzik, Tomasz
Raman, Venkatesh
Rescigno, Adele

Sakoda, Genki
Satti, Srinivasa Rao
Stamoulis, Georgios
Sun, Zhaohong
T. P., Sandhya
Vialette, Stéphane
Williams, Aaron
Wyels, Cynthia
Zhang, Melvin
Zhou, Yi

Invited Talks

Some Recent New Directions in Multivariate Algorithmics

Michael Fellows

Department of Informatics, University of Bergen, Norway
Michael.Fellows@uib.no

Abstract. The talk will try to do three things:

(1) Give a basic introduction to the key ideas of parameterized complexity/multivariate algorithmics, for those who may be unfamiliar with this area of research. The account will be somewhat idiosyncratic, colorful and concrete, and may offer some new perspectives even to those who are conversant in the technical ideas of this area.
(2) Briefly survey some of the key achievements of this area of research so far, and the major themes, such as the equivalency between P-time kernelization and FPT that have emerged.
(3) Exposit recent research directions in this area that have attracted substantial new research funding in various countries of the world.

Survey of Some Recent Near Polynomial Time Results for Parity Games

Sanjay Jain

Department of Computer Science, National University of Singapore,
13 Computing Drive, COM1, Singapore 117417, Republic of Singapore
{sanjay}@comp.nus.edu.sg

Abstract. In this talk we will describe a Quasi Polynomial time algorithm for parity games given by Calude et al (STOC 2017). The runtime for the algorithm is $O(n^{\log(m)+6})$, where n is the number of nodes and m is the number of colours (priorities). The parameterised parity game – with n nodes and m distinct colours is proven to be in the class of fixed parameter tractable problems (FPT) when parameterised over m. The corresponding runtime is $O(n^5 + g(m))$, where $g(m)$ can be taken to be m^{m+6}. We will also discuss the next developments in the field which improved the above algorithm by making it simultaneously in near linear space by Jurdzinski and Lazic (LICS 2017) and Fearnley et al (SPIN 2017). Recently, Lehtinen (LICS 2018) introduced the notion of register index complexity and showed that this is logarithmic in the number of nodes; furthermore, a game with register index complexity k, the parity game can be solved in time $m^{O(k)} \cdot n^{O(1)}$ which provides another quasipolynomial time algorithm for parity games.

S. Jain was supported in part by the Singapore Ministry of Education Academic Research Fund Tier 2 grant MOE2016-T2-1-019/R146-000-234-112 and NUS grant C252-000-087-001.

Range Minimum Queries and Applications

Kunihiko Sadakane

Department of Mathematical Informatics, Graduate School of Information Science and Technology, The University of Tokyo
sada@mist.i.u-tokyo.ac.jp

Consider the following problem.

Range Minimum Query, RMQ Given an array $A[1..n]$ and a range $[s,t] \subset [1,n]$, a range minimum query asks the position of the minimum value in $A[s..t]$. If there exist more than one minimum values in the query range, return the leftmost one.

We consider the indexing problem, that is, given the array A, we first construct a data structure D_A, then given a query range, we solve the problem using D_A. There exists a linear space ($O(n)$ words) data structure for the RMQ problem supporting constant time queries [3, 4]. It is however complicated and there have been no efficient implementations until recently. In 2000, a simple solution [1] was given and after that, constant query time RMQ data structures are used in many algorithms.

In this talk, we explain an $O(n)$-word data structure for the RMQ problem. Then we reduce the size of the data structure to just $2n + o(n)$ bits [2]. We also explain applications of the problem such as compressed suffix trees [5].

References

1. 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)
2. Fischer, J., Heun, V.: Space-efficient preprocessing schemes for range minimum queries on static arrays. SIAM J. Comput. **40**(2), 465–492 (2011)
3. Gabow, H.N., Bentley, J.L., Tarjan, R.E.: Scaling and related techniques for geometry problems. In: Proceedings of ACM Symposium on Theory of Computing (STOC), pp. 135–143, New York, USA. ACM Press (1984)
4. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. **13**(2), 338–355 (1984)
5. Sadakane, K.: Compressed suffix trees with full functionality. Theory Comput. Syst. **41**(4), 589–607 (2007)

Contents

Collision-Free Routing Problem with Restricted L-Path

Jammigumpula Ajay(✉) and Sasanka Roy

Indian Statistical Institute, Kolkata 700108, India
ak.jammi@gmail.com, sasanka.ro@gmail.com

Abstract. We consider a variant of collision-free routing problem CRP. In this problem, we are given set C of n vehicles which are moving in a plane along a predefined directed rectilinear path. Our objective (CRP) is to find the maximum number of vehicles that can move without collision. CRP is shown to be NP-Hard by Ajaykumar et al. [1]. It was also shown that the approximation of this problem is as hard as Maximum Independent Set problem (MIS) even if the paths between a pair of vehicles intersects at most once. So we study the constrained version $CCRP$ of CRP in which each vehicle c_i is allowed to move in a directed L-Shaped Path.

We prove $CCRP$ is NP-Hard by a reduction from MIS in L-graphs, which was proved to be NP-Hard even for unit L-graph by Lahiri et al. [2]. Simultaneously, we show that any $CCRP$ can be partitioned into collection $\mathcal{L}$ of L-graphs such that $CCRP$ reduces to a problem of finding MIS in L-graph for each partition in $\mathcal{L}$. Thus we show that any algorithm, that can produce a β-approximation for L-graph, would produce a β-approximation for $CCRP$. We show that unit L-graphs intersected by an axis-parallel line is Co-comparable. For this problem, we propose an algorithm for finding MIS that runs in $O(n^2)$ time and uses $O(n)$ space. As a corollary, we get a 2-approximation algorithm for finding MIS of unit L-graph that runs in $O(n^2)$ time and uses $O(n)$ space.

Keywords: Maximum Independent Set · L-Graphs
Approximation algorithm · Collision-free · Co-comparable graph

1 Introduction

The problem is motivated by the recent development of automated driver-less vehicles, which are capable of various decision activities such as motion-controlling, path planning. If we consider a simple road network like Manhattan and restrict it to be one way for the simplicity of driver-less vehicles routes, many interesting problems can be seen in this network.

The paper on Problems on One Way Road Networks [1], gives an idea of One Way Road Network ($OWRN$) and Traffic Configuration (TC), where each vehicle moves in a predetermined path in an $OWRN$ and the aim is to find the maximum number of vehicles that can be allowed to move without having any

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 1–13, 2018.
https://doi.org/10.1007/978-3-319-94667-2_1

collision for a given TC. They proved that this problem is NP-hard by reducing it to MIS, and also showed that the approximation for this problem is as hard as approximating MIS. It is known that, for every fixed $\epsilon > 0$, MIS cannot be approximated within a multiplicative factor of $n^{1-\epsilon}$ for a general graph, unless $NP = ZPP$ [4].

We can generalize TC to CRP, where each vehicle is allowed to move in a rectilinear path, replacing the vertices of the $OWRN$ by their coordinate points and the path same as in TC. Similar kind of road network has also been studied by Dasler and Mount [3].

It is easy to see if we constrain the vehicles to move in directed straight lines parallel to the axis, then the corresponding graph to CRP will be a Bipartite Graph. MIS of a Bipartite Graph can be computed using Kőnig's Theorem [7] and Network-Flow Algorithm [8] in polynomial-time.

Our Contribution:
We considered a special case of CRP, called constrained collision-free routing problem $CCRP$, where each vehicle is restricted to move in an L-shaped path. We prove $CCRP$ is NP-hard by reduction from MIS in L-graphs.

Simultaneously, we show that any $CCRP$ can be partitioned into a collection $\mathcal{L}$ of L-graphs such that $CCRP$ reduces to a problem of finding MIS for each partition in $\mathcal{L}$. Thus we show that any algorithm, that can produce a β-approximation for L-graph, would produce a β-approximation for $CCRP$. Since the best-known algorithm for L-graph by Lahiri, Mukherjee, and Subramanian [2] has $O(\log^2 n)$-approximation, $CCRP$ has $O(\log^2 n)$-approximation.

Further, we extended our work to study the properties of unit L-graph[1], denoted as G_{LU}, where all the objects are of unit size. We prove that unit L-graph, denoted as $G_{LU}(\ell)$, where all L's are intersected by a single axis parallel line ℓ is a Co-comparable graph. This characterization gives us an algorithm for finding MIS in $O(n^2)$ time using $O(n^2)$ space using results by Rose, Tarjan and Lueker [9]. We propose a dynamic programming based algorithm for finding MIS of $G_{LU}(\ell)$ that runs in $O(n^2)$ time and uses $O(n)$ space. Also as a corollary, we get a 2-approximation for finding MIS of G_{LU}. The following are few definitions and notations we will use throughout this work.

Definition 1. *An L-shaped path $P_i = (p_i, q_i, r_i)$ is defined by three co-ordinate points, where the path segment p_iq_i of P_i forms a vertical segment (directed downwards) and path segment q_ir_i of P_i forms a horizontal segment (directed rightwards).*

Definition 2. *A vehicle c_i is defined as a 3-tuple (t_i, s_i, P_i), where t_i is the start time, s_i is a constant speed with which it will travel till it reaches the destination, P_i is the L-shaped path (with source p_i and destination r_i).*

Definition 3. *If two L-shaped paths have a common point, then they are said to be intersecting with each other. This common point is called the intersection point of the two vehicles moving in these L-shaped paths.*

[1] If both the horizontal and vertical segments of an L are of unit length then we call it a unit L.

Definition 4. *If two vehicles reach an intersection point orthogonally at the same time, then we call it a collision.*

2 Hardness of *CCRP*

In this section we prove the hardness of *CCRP*. Throughout this paper, we assume that each vehicle is moving with a unit velocity, and the paths intersect at a single point.

Definition 5. *We define $x(p)$, $y(p)$ as the X-coordinate and Y-coordinate of the point p.*

Observation 1 *If two paths P_i and P_j intersect with each other such that $x(q_i) < x(q_j)$, then $y(q_i) > y(q_j)$.*

Lemma 1. *If two vehicles collide with each other, then a third vehicle whose path intersects with both paths would either (i) collide with both the vehicles or (ii) does not collide with both the vehicles.*

Proof. Consider three vehicles c_1, c_2 and c_3 with paths P_1, P_2 and P_3, respectively. Without loss of generality, we can assume $x(q_1) < x(q_2) < x(q_3)$. Thus from Observation 1 we can claim $y(q_1) > y(q_2) > y(q_3)$. Let P_1, P_2 intersect at point γ, P_3 intersects with both P_1 and P_2 at points α, β respectively. Let the distance from γ to α be a units, and the distance from α to β be b units, refer to Fig. 1.

Let c_1 reaches point γ at time t^1_γ, then the time at which it reaches point α is $t^1_\alpha = t^1_\gamma + a$. Let c_2 reaches point γ at time t^2_γ, then the time at which it will reach point β is $t^2_\beta = t^2_\gamma + a + b$. Let c_3 reaches point α at time t^3_α, then the time at which c_3 reaches point β is $t^3_\beta = t^3_\alpha + b$.

Clearly $(t^1_\alpha - t^1_\gamma) - (t^2_\beta - t^2_\gamma) + (t^3_\beta - t^3_\alpha) = 0$, rearranging the terms we get, $(t^1_\alpha - t^3_\alpha) + (t^2_\gamma - t^1_\gamma) + (t^3_\beta - t^2_\beta) = 0$. If two vehicles collide then one of the three parts in the above equation will become zero. Thus, if one of the remaining two parts becomes zero so does the other, this concludes the proof.

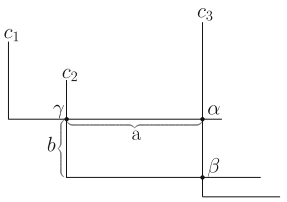


Fig. 1. Illustration of Lemma 1

Definition 6. *We define an L-path graph G_L^t as a collision graph of vehicles moving in an L-shaped path, where each vehicle represents a vertex in G_L^t, and there is an edge between two vertices in G_L^t if the respective vehicles collide.*

So our $CCRP$ problem reduces to the problem of finding MIS of G_L^t. We may use MIS of G_L^t and $CCRP$ interchangeably. We denote $|S|$ as the cardinality of the set S. We also denote $|a - b|$ as the distance between two points a and b on a real line.

Definition 7. *Any induced sub-graph H^t of G_L^t is called a connected component in L-path graph, if for every vertex pair u, v in H^t there exists a path from u to v in H^t.*

Theorem 1. *If the path of a vehicle c_i intersect with paths of two or more vehicles in a connected component and it collides with one of them, then it collides with all the vehicles whose path it intersects.*

Proof. We prove this theorem using strong induction. As the base case, if the connected component has two vehicles and the path of a third vehicle intersects the path of both vehicles, and it collides with one of them, then from Lemma 1 the statement holds for the base case of three vehicles.

We assume that any connected component of size less than k follows this property, and we prove the claim holds for any connected component of size k.

Given any connected component H^t of size k, select any vehicle c_3, if its path intersects with only one vehicle (which is a collision since c_3 belongs to the connected component), then the claim is true. If the path of c_3 intersects with the path of more than one vehicle, then it must collide with at least one of the vehicles since it belongs to the connected component. So we choose one intersection and one collision to prove that the intersection will be a collision, thus inductively prove that all intersections will be collisions.

Let c_1 and c_2 be vehicles such that either c_1 or c_2 has a collision with c_3, while the other has an intersection with the path of c_3. Without loss of generality we can assume $y(q_1) > y(q_2)$.

Delete c_3 from H^t and find the path in H^t with minimum number of nodes from corresponding vertex of c_1 to respective vertex of c_2. Consider all the corresponding vehicles of the vertices in this path and remove the rest of the vehicles. If c_1 and c_2 intersect with each other then by our inductive assumption c_1 and c_2 belong to a connected component of size less than k. Thus c_1 and c_2 collide with each other. By Lemma 1 c_3 collides with both c_1 and c_2.

Thus we only need to show for the case where P_1 and P_2 doesn't intersect with each other. Since it is the shortest path in H^t no vehicle's path will intersect more than two vehicles. P_1 and P_2 intersect with only one path each. Note all these vehicles together form a single connected component. If we insert c_3 it will still collide with c_1 (or c_2) while its path intersect with the path of c_2 (or c_1).

Here we have following two cases, where in each case we replace c_1 with another vehicle c_1' and c_2 with another vehicle c_2'. Such that (i) P_1' and P_2' will

intersect and (ii) the set of vehicles in the plane after replacing c_1 and c_2 will still be a connected component.

Case 1. $x(q_1) < x(q_2)$. Since we assumed $y(q_1) > y(q_2)$ and P_1 and P_2 doesn't intersect, we can have following three configurations as shown in Fig. 2.

For configuration in Fig. 2(a) If we extend q_1r_1 in rightward direction and p_2q_2 in upward direction they intersect as shown in Fig. 2(d) where c_1' and c_2' represent this modification.

For configuration in Fig. 2(b) If we extend p_2q_2 in upward direction they intersect as shown in Fig. 2(e) where c_1' and c_2' represent this modification.

For configuration in Fig. 2(c) If we extend q_1r_1 in rightward direction they intersect as shown in Fig. 2(f) where c_1' and c_2' represent this modification.

We replace c_1 with c_1' and c_2 with c_2', such that $t_1' = t_1$, $p_1' = p_1$, $q_1' = q_1$, $y(r_1') = y(r_1)$, $x(r_1') = max(x(r_1), x(q_2)+\epsilon)$, and $r_2' = r_2$, $q_2' = q_2$, $x(p_2') = x(p_2)$, $y(p_2') = max(y(p_2), y(q_1) + \epsilon)$, $t_2' = t_2 - (y(p_2') - y(p_2))$, for some $\epsilon > 0$.

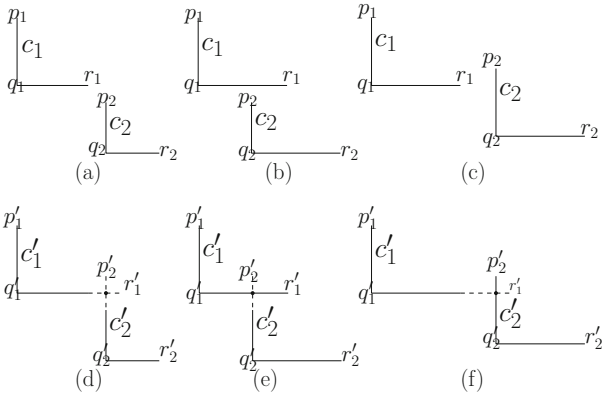


Fig. 2. Illustration of Case 1

The above modification doesn't change the time at which c_1' (or c_2') reaches the collision point of c_1 (or c_2). Hence, after the replacement, c_1' and c_2' belongs to the same connected component. In our construction we also made sure that P_1' and P_2' intersect. Now we have a connected component of size less than k. Hence c_1' and c_2' must also collide.

Now consider c_3, if it collides with c_1 (or c_2) then it must also collide with c_1' (or c_2') according to our construction. From Lemma 1 it is evident that it collides with both c_1' and c_2'. Hence the intersection must also be a collision.

Case 2. $x(q_1) > x(q_2)$. In the previous case we only extended one of the line segments for c_1 and c_2 to get c_1' and c_2' respectively, but in this case we are moving the segment i.e. both points p_1, q_1 are moved by some distance leftwards or both q_1, r_1 are moved by some distance downwards. In order to keep the

connectivity we check the immediate neighbour c_4 of c_1 and the segment say p_1q_1 (or q_1r_1) of P_1 with which the path P_4 intersects. Then modify the other segment q_1r_1 (or p_1q_1) of P_1 to get c_1'. c_2' can be generated just by extending one of the segments. The vehicle c_4 that collides with c_1 could have $y(q_4) > y(q_1)$ or $y(q_4) < y(q_1)$.

1. If $y(q_4) < y(q_1)$ (i.e., P_4 intersects segment q_1r_1 of P_1) then we can have following three configurations as shown in Fig. 3(a),(b),(c).
   For configuration in Fig. 3(a) If we shift p_1, q_1 to the left direction and extend p_2q_2 in upward direction and P_3 intersects segments q_1r_1 and q_2r_2 then they intersect as shown in Fig. 2(d) where c_1' and c_2' represent this modification.
   For configuration in Fig. 3(b) If we shift p_1, q_1 to the left direction and P_3 intersects segments q_1r_1 and q_2r_2 then they intersect as shown in Fig. 2(e) where c_1' and c_2' represent this modification.
   For configuration in Fig. 3(c) If we shift p_1, q_1 to the left direction and P_3 intersects segments p_1q_1 and p_2q_2 then they intersect as shown in Fig. 2(f) where c_1' and c_2' represent this modification.
   We replace c_1 with c_1', c_2 with c_2' such that, $y(p_1') = y(p_1)$, $x(p_1') = x(p_2) - \epsilon$, $y(q_1') = y(q_1)$, $x(q_1') = x(p_1')$, $t_1' = t_1 - (x(q_1') - x(q_1))$, $r_1' = r_1$ and $y(p_2') = max(y(p_2), y(p_1) + \epsilon)$, $r_2' = r_2$, $q_2' = q_2$, $t_2' = t_2 - (y(p_2') - y(p_2))$, for some $\epsilon > 0$.
   By our construction P_1' and P_2' intersect with each other, c_1' collides with c_4, and c_2' reaches the collision points at the same time as c_2. Hence even after replacing c_1 by c_1' and c_2 with c_2', the whole component remains connected with size less than k. By inductive hypothesis c_1' collides with c_2'.
   Now we have the following three scenarios, (a), (b), (c) as shown in Fig. 3 for scenario (a) and (b), from Lemma 1, it is evident that c_3 collides with both c_1' and c_2' as shown in figure (d) and (e). Hence it collides with both c_1 and c_2 as well.
   In Fig. 3(c) if c_3 collides with c_1, then it must also collide with c_1' which can be proved in a way similar to Lemma 1 by considering c_1, c_1' and c_3. Since c_1' and c_2' collide with each other c_3 must also collide with c_2'. Hence c_3 collides with both c_1 and c_2. Else, if c_3 collides with c_2 then it trivially collides with c_2'. Hence c_3 collides with c_1'. Thus c_3 collides c_1 which can be proved in a way similar to Lemma 1 by considering c_1, c_1' and c_3.
2. If $y(q_4) > y(q_1)$ (i.e., P_4 intersects segment p_1q_1 of P_1) then similar arguments can be made but instead of shifting the vertical segment p_1r_1 by some distance, we shift the horizontal segment q_1r_1. This makes sure that replacing c_1 and c_2 with c_1' and c_2' respectively doesn't disturb the connectedness. We can have the following three configurations as shown in Fig. 4.
   For configuration in Fig. 4(a) If we shift q_1, r_1 to downward direction and P_3 intersects segments p_1q_1 and p_2q_2 then they intersect as shown in Fig. 2(d) where c_1' and c_2' represent this modification.

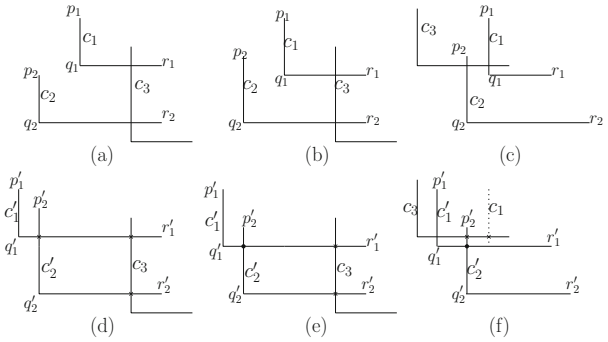


Fig. 3. Illustration of Case 2.1

For configuration in Fig. 4(b) If we shift q_1, r_1 to downward direction and extend q_2r_2 in rightward direction and P_3 intersects segments p_1q_1 and p_2q_2 then they intersect as shown in Fig. 2(e) where c'_1 and c'_2 represent this modification.
For configuration in Fig. 4(c) If we shift q_1, r_1 to downward direction and P_3 intersects segments q_1r_1 and q_2r_2 then they intersect as shown in Fig. 2(f) where c'_1 and c'_2 represent this modification.
Replace c_1 with c'_1, c_2 with c'_2 such that $p'_1 = p_1$, $x(q'_1) = x(q_1)$, $y(q'_1) = y(q_2) - \epsilon$, $x(r'_1) = x(r_1)$, $y(r'_1) = y(q'_1)$, $t'_1 = t_1$ and $p'_2 = p_2$, $q'_2 = q_2$, $x(r'_2) = max(x(r_2), x(q_1) + \epsilon)$, $y(r'_2) = y(r_2)$, $t'_2 = t_2$, for some $\epsilon > 0$. The proof can be argued in a similar manner to the above sub-case.

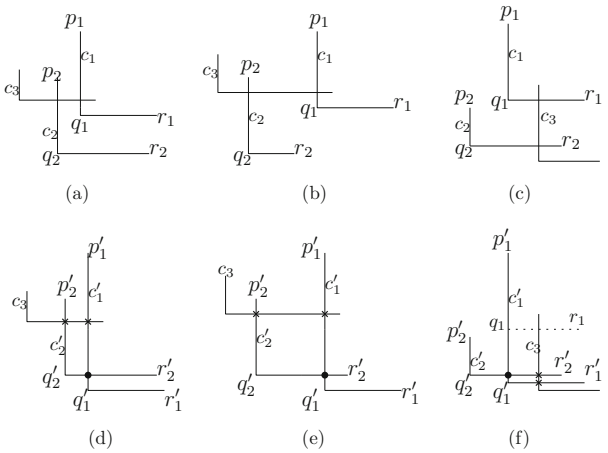


Fig. 4. Illustration of Case 2.2

Definition 8. *We define an L-graph G_L as an intersection graph of L-shaped paths, where each L-shaped path represents a vertex in G_L, and there is an edge between two vertices in G_L if the respective L-shaped paths intersect.*

Now we propose an algorithm to reduce any given instance of G_L to an instance of G_L^t, as follows: For every object $\ell \in G_L$ there exists a vehicle $c \in G_L^t$, such that if and only if $l_i, l_j \in G_L$ has an edge then their corresponding vehicles c_i and c_j collides in G_L^t.

Algorithm 1. Assignment of Time in G_L to obtain G_L^t

```
1: procedure ASSIGNTIME(C, S, i)
2:     insert i into S
3:     for ∀j ∈ C do
4:         if i = 0 and j ∉ S then
5:             set t_j = 0
6:         else
7:             if j ∉ S and intersects with i then
8:                 SETTIME(C, i, j)
9:                 ASSIGNTIME(C, S, j)
10:            end if
11:        end if
12:    end for
13: end procedure
```

Theorem 2. *Given an L-graph, there exists a G_L^t Computable in polynomial time, such that the cardinality of MIS of G_L is k if and only if the cardinality of MIS of G_L^t is k.*

Proof. For each object l_i in L-graph, assign a vehicle c_i with path as l_i and a unit velocity. Insert all vehicles into set C. Let S be an empty set. Now call the procedure ASSIGNTIME$(C, S, 0)$. This will give a time assignment to each and every vehicle. The procedure SETTIME(C, i, j) assigns time t_j such that c_j will collide with c_i (i.e. if l_i, l_j intersect at point g then $t_j = t_i + |x(p_i) - x(g)| + |y(p_i) - y(g)| - |x(p_j) - x(g)| - |y(p_j) - y(g)|$).

In the above assignment for each connected component, the time of one of the vehicle is set to zero and every other vehicle is set to collide with at least one of the vehicles in the connected component. Hence from Theorem 1 we have, every intersection in G_L as a collision in G_L^t.

This assignment might assign negative time to some vehicles. To ensure that the start time to be non-negative for each vehicle, find the minimum time assignment out of all vehicles and subtract that value from the time of each vehicle.

Algorithm 2. Procedure to partition G_L^t

```
1: procedure SEPERATESET(i)
2:     U = UniversalSet, S = φ
3:     insert i into S
4:     for ∀j ∈ U do
5:         if j ∉ S and collides with i then
6:             insert j into S
7:             insert SEPERATESET(j) into S
8:         end if
9:     end for
10:    return S
11: end procedure
```

3 Approximation for MIS of G_L^t

We propose an algorithm to partition the G_L^t to collections of G_L's.

Lemma 2. *Any set S generated by the procedure* SEPERATESET *is independent of the set $U \setminus S$, i.e.* $\text{MIS}(U) = \text{MIS}(S) + \text{MIS}(U \setminus S)$.

Lemma 3. *Any set S generated by above algorithm is an L-Graph (G_L).*

Theorem 3. *For any L-path graph, there exists an approximation factor equivalent to L-graph. i.e. there exists a $O(\log^2 n)$ approximation algorithm.*

Proof. From Lemma 2 and Lemma 3, it is evident that given any L-path graph, we can separate the L-path graph into subsets $S_1, S_2, \ldots$, and all of them are pair wise independent (i.e. no collision between objects from two different sets) and from Theorem 1 each set S_i can be treated as an L-graph i.e., each intersection of objects belonging to same set S_i is nothing but a collision in S_i.

Now apply the known approximation algorithm of L-graphs [2] for each S_i, and return the union. Let $\mathcal{O}pt(S_i)$ denote the optimal solution for S_i and $\mathcal{S}ol(S_i)$ denote the solution generated by the algorithm [2]. Since we know $\mathcal{O}pt(S_i) \leq (k \log^2 n)\mathcal{S}ol(S_i)$, summing over all the sets on both sides will result in the desired inequality, $\sum_{i=1} \mathcal{O}pt(S_i) \leq \sum_{i=1}(k \log^2 n)\mathcal{S}ol(S_i)$. This concludes the proof.

4 Unit L Graph Approximation

Definition 9. *A unit L-graph G_{LU} is a special graph of G_L where each L-shaped path is of the unit size, i.e., both the horizontal and vertical segments are of unit length each.*

In this section, we design a 2-approximation algorithm for the maximum independent set in a unit L-graph problem. Let $S = \{P_1, P_2, \ldots, P_n\}$ be a set of n unit L-shaped paths in a plane. We first place vertical lines from leftmost to

rightmost with a unit distance between each consecutive pair of lines. Assume that there are k such vertical lines $\{L_1, L_2, \ldots, L_k\}$. Let $S_i \subseteq S$ be the set of L-shaped paths intersected by the line L_i. The idea is to find MIS for each S_i and then combine them to produce an approximate solution. This method is well known for finding an approximate solution for MIS of fixed height rectangle by Agarwal et al. [5] and for the unit disk by Nandy et al. [6]. But our problem is different in a sense that the intersection graph $I(S_i)$ of a S_i may not be a triangulated graph. We can construct an $I(S_i)$ that contains a four-cycle as shown in Fig. 5. So we show that $I(S_i)$ is a co-comparable graph. Then we give a dynamic programming based algorithm that solves MIS of $I(S_i)$ in $O(n^2)$ time using $O(n)$ space.

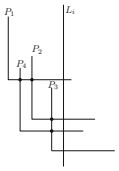


Fig. 5. G_{LU} with four cycle

Observation 2 *Any two L-shaped paths, $P_a \in S_i$ and $P_b \in S_i$ are independent if $|y(q_a) - y(q_b)| > 1$, for $1 \leq i \leq k$.*

Observation 3 *Any two L-shaped paths, $P_a \in S_i$ and $P_b \in S_i$ with $y(q_a) < y(q_b)$ are independent if $x(q_a) < x(q_b)$.*

Observation 4 *Any two L-shaped paths, $P_a \in S_i$ and $P_b \in S_j$ are independent if $|i - j| > 1$, for $1 \leq i, j \leq k$.*

Lemma 4. *If P_1, P_2, P_3 are three unit L-shaped paths that intersect a vertical line L_i such that (i) $y(q_1) > y(q_2) > y(q_3)$, (ii) P_1, P_2 doesn't intersect and (iii) P_2, P_3 doesn't intersect, then P_1, P_3 doesn't intersect.*

Definition 10. *We denote $\widetilde{G} = (V, \widetilde{E})$ as the complimentary graph of $G = (V, E)$, such that $(u, v) \in \widetilde{E}$ if and only if $(u, v) \notin E$, for all $u, v \in V$.*

Lemma 5. *The graph $\widetilde{G}_{LU}$ of the unit L-shaped path intersecting a vertical line L_i is a Comparable graph.*

Proof. We show that $\widetilde{G}_{LU}$ is orientable, such that if there is a directed edge from vertex a to vertex b and there is a directed edge from vertex b to vertex c, then there is a directed edge from vertex a to vertex c, for all vertices $a \neq b \neq c$ in the $\widetilde{G}_{LU}$.

The ordering of vertices is as follows: A vertex a precedes a vertex b if the Y-coordinates of the respective L-shaped paths P_a and P_b follow the inequality

$y(q_a) > y(q_b)$. Now if there is an edge between any two vertices a and b in the graph $\widetilde{G}_{LU}$ and a precedes b then direct the edge from a to b.

In the above mentioned ordering, we can conclude that $\widetilde{G}_{LU}$ is a comparable graph because if and only if there is an edge between a, b, and b,c in $\widetilde{G}_{LU}$ then a,b and b,c are independent in G_{LU}. Since they are in increasing order, by Lemma 4 a, c is also independent in G_{LU}. Thus there is an edge between a and c in $\widetilde{G}_{LU}$. This proves the lemma.

Corollary 1. *The graph $G_{LU}(L_i)$ formed by unit L-shaped paths which are intersecting with a vertical line L_i is a Co-comparable graph i.e. the graph $G_{LU}(L_i)$ formed by S_i is Co-comparable.*

Given any S_i, we sort the elements based on their Y-coordinates. i.e., a path P_a will have an index less than P_b if $y(q_a) < y(q_b)$. For the sake of simplicity we refer to the path at index k as P_k.

For any index k, let $R(k)$ be the maximum possible independent set till k that includes the path P_k and let $J_k = \{P_{j_1}, P_{j_2}, \ldots, P_{j_l}\}$ be the set of all paths that doesn't intersect with P_k and have index less than k.

Observation 5 $R(k) = \begin{cases} 1 & \text{if } J_k = \phi \\ 1 + max(R(j_1), R(j_2), \ldots, R(j_l)) & \text{otherwise} \end{cases}$

Algorithm 3. Computing $R(k)$ for each index in S_i

```
1: procedure LINEINTERSECTMIS(S_i)
2:     R, B are arrays of size |S_i|
3:     for k = 1 to |S_i| do
4:         Set R(k) = 0, B(k) = −1
5:     end for
6:     R(1) = 1
7:     for k = 2 to |S_i| do
8:         for j = 1 to k − 1 do
9:             if P_k and P_j doesn't intersect then
10:                if R(j) > R(k) then
11:                    R(k) = R(j)
12:                    B(k) = j
13:                end if
14:            end if
15:        end for
16:        R(k) = R(k) + 1
17:    end for
18:    return R, B
19: end procedure
```

Lemma 6. *The recurrence to compute the maximum independent set in S_i till index k is $MIS(k) = max(MIS(k-1), R(k))$.*

If we compute $R(k)$ for all index k, then in a single run i.e. $O(|S_i|)$ we can compute the maximum independent set for S_i.

Note that, the procedure LINEINTERSECTMIS(S_i) can be modified to solves MIS for S_i optimally. Run LINEINTERSECTMIS on each S_i, for $1 \leq i \leq k$ and let E_i be the maximum independent in S_i. We define two sets $Even_{OPT} = \bigcup_{1 \leq i \leq k,\ i\ is\ even} E_i$ and $Odd_{OPT} = \bigcup_{1 \leq i \leq k,\ i\ is\ odd} E_i$. We report the set with the maximum cardinality among $Even_{OPT}$ and Odd_{OPT} as the result of our algorithm. Thus we have the following theorem.

Theorem 4. *Our algorithm produces a 2-approximation for MIS in G_{LU}, with a time complexity of $O(n^2)$ and a space complexity of $O(n)$.*

5 Conclusion

We obtained hardness results and approximation algorithm for *CCRP*. We showed that $G_{LU}(\ell)$ is a Co-comparable graph. We proposed a dynamic programming based algorithm for finding MIS of $G_{LU}(\ell)$ in $O(n^2)$ time using linear space. Which produces 2-approximation for finding MIS of G_{LU} with $O(n^2)$ time and $O(n)$ space complexity. Finally we pose the following open problems:

1. Can a 2-approximation for MIS of G_{LU} be obtained in sub-quadratic time?
2. Does there exist a polynomial time sub-linear approximation algorithm for *CRP* when the vehicles are moving only along XY-monotone paths?

Acknowledgments. The authors are thankful to Joydeep Mukherjee for many useful discussions.

References

1. Ajaykumar, J., Das, A., Saikia, N., Karmakar, A.: Problems on one way road networks. In: CCCG 2016, pp. 303–308 (2016)
2. Lahiri, A., Mukherjee, J., Subramanian, C.R.: Maximum independent set on B_1-VPG graphs. In: Lu, Z., Kim, D., Wu, W., Li, W., Du, D.-Z. (eds.) COCOA 2015. LNCS, vol. 9486, pp. 633–646. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26626-8_46
3. Dasler, P., Mount, D.M.: On the complexity of an unregulated traffic crossing. In: Dehne, F., Sack, J.-R., Stege, U. (eds.) WADS 2015. LNCS, vol. 9214, pp. 224–235. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21840-3_19
4. Johan, H.: Clique is hard to approximate within $n^{1-\epsilon}$. Acta Mathematica **182**(1), 105–142 (1999)
5. Agarwal, P.K., van Kerveld, M., Suri, S.: Label placement by maximum independent set in rectangles. Comput. Geom. **11**(3), 209–218 (1998)

6. Nandy, S.C., Pandit, S., Roy, S.: Faster approximation for maximum independent set on unit disk graph. Inf. Process. Lett. **127**, 58–61 (2017)
7. Kőnig, D.: Gráfok és mátrixok. Matematikai és Fizikai Lapok **38**, 116–119 (1931)
8. Malhotra, V.M., Pramodh Kumar, M., Maheshwari, S.N.: An $O(|V|^3)$ algorithm for finding maximum flows in networks. Inf. Process. Lett. **7**(6), 277–278 (1978)
9. Rose, D.J., Tarjan, R.E., Lueker, G.S.: Algorithmic aspects of vertex elimination on graphs. SIAM J. Comput. **5**(2), 266–283 (1976)

Linear Clique-Width of Bi-complement Reducible Graphs

Bogdan Alecu[1], Vadim Lozin[1(✉)], and Viktor Zamaraev[2]

[1] Mathematics Institute, University of Warwick, Coventry CV4 7AL, UK
{B.Alecu,V.Lozin}@warwick.ac.uk
[2] Department of Computer Science, Durham University,
South Road, Durham DH1 3LE, UK
viktor.zamaraev@gmail.com

Abstract. We prove that in the class of bi-complement reducible graphs linear clique-width is unbounded and show that this class contains exactly two minimal hereditary subclasses of unbounded linear clique-width.

1 Introduction

Clique-width is a graph parameter which is of primary importance in algorithmic graph theory, because many problems in this area that are generally NP-hard can be solved efficiently when restricted to graphs of bounded clique-width. This parameter generalizes tree-width in the sense that bounded tree-width implies bounded clique-width but not necessarily vice versa.

Recently, many classes of graphs have been shown to be of bounded clique-width, and for many others, the clique-width was shown to be unbounded, see e.g. [2,5,13,14]. Most of these studies concern hereditary classes, i.e. classes closed under taking induced subgraphs. This restriction is justified by the fact that the clique-width of a graph G can never be smaller than the clique-width of an induced subgraph of G. An important feature of hereditary classes is that they admit a description in terms of minimal forbidden induced subgraphs, i.e. minimal graphs that do not belong to the class.

In a similar way, in the study of clique-width of particular importance are *minimal* classes of graphs of unbounded clique-width. The first two hereditary classes of this type have been identified in [12] and only recently it was shown in [4] that the number of such classes is infinite. What is interesting is that all the classes found in [4] are also minimal hereditary classes of unbounded *linear* clique-width.

Linear clique-width is a restricted version of clique-width and the relationship between these two parameters is similar to the relationship between tree-width and path-width. The notion of linear clique-width became an important ingredient in the proof of hardness of computing clique-width [6] and received considerable attention in recent years in the literature [1,3,9–11]. Nevertheless, our knowledge of this parameter is still restricted. In particular, we know very

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 14–25, 2018.
https://doi.org/10.1007/978-3-319-94667-2_2

little about the behaviour of this parameter on graphs of bounded clique-width. In this respect, the recent paper [3] is of particular interest. It deals with the class of complement reducible graphs, also known as cographs, where the clique-width is known to be bounded, while the linear clique-width is not [9]. The authors of [3] show that there exist precisely two minimal hereditary subclasses of cographs of unbounded linear clique-width. These two subclasses are complement to each other and are known in the literature under various names, such as trivially perfect [8] or quasi-threshold graphs [15].

In the present paper, we study a bipartite analog of cographs, known as *bi-complement reducible graphs* [7] (*bi-cographs* for short), where the clique-width also is known to be bounded. We prove that the linear clique-width is unbounded in the class of bi-cographs and, similarly to [3], show that there exist precisely two minimal hereditary subclasses of bi-cographs of unbounded linear clique-width. However, our solution differs from that in [3] in two important aspects.

Firstly, the two classes we discover in this paper had never been studied before and are of independent interest. We characterize them in terms of minimal forbidden induced subgraphs.

Secondly, and most importantly, we develop an entirely new approach to prove our results. In particular, to prove unboundedness of clique-width we introduce an auxiliary graph parameter which bounds the linear clique-width from below and provides a more flexible tool to prove results of this type. To show minimality of our classes we develop a straightforward approach, which avoids the notion of well-quasi-ordering used by the authors of [3]. Their approach is applicable only to classes well-quasi-ordered by induced subgraphs, which is not the case, for instance, for trees, where clique-width is bounded and linear clique-width is not, similarly to cographs and bi-cographs.

The organization of the paper is as follows. All preliminary information related to the topic of the paper can be found in Sect. 2. Then in Sect. 3 we introduce two subclasses of bi-cographs and characterize them in terms of minimal forbidden induced subgraphs. Section 4 is devoted to the proof of unboundedness of linear clique-width in both subclasses and Sect. 5 is devoted to the proof of their minimality. Section 6 concludes the paper with a number of open problems.

2 Preliminaries

This section introduces basic terminology and notation used in the paper.

2.1 Graphs

Throughout the paper, we will be working with undirected graphs, with no loops or multiple edges. The vertex set and the edge set of a graph G are denoted by $V(G)$ and $E(G)$, respectively. For a vertex $x \in V(G)$ we denote by $N(x)$ the neighbourhood of x, i.e. the set of vertices of G adjacent to x. A subgraph of G induced by a subset of vertices $U \subseteq V(G)$ is denoted $G[U]$. We use the following notation for specific graphs:

- P_n is the chordless path on n vertices.
- C_n is the chordless cycle on n vertices.
- The domino is the graph obtained from a C_6 by adding an edge between one pair of antipodal vertices.
- Sun_n is the graph obtained from a C_n by adding a pendant vertex to each vertex of the cycle.
- Star_{ijk} is the tree with exactly 3 leaves at distances i, j and k from the only vertex of degree 3.

We will be studying bipartite graphs in particular. We will distinguish between *coloured* and *uncoloured* classes of bipartite graphs. In a coloured class, all bipartite graphs come with a bipartition of their vertex set into two independent sets B and W that we will refer to as *black* and *white* vertices, respectively.

For a coloured bipartite graph $G = (B, W, E)$, we define the *bipartite complement* of G to be the coloured bipartite graph $\widetilde{G} = (B, W, E')$, where for any two vertices $x \in B$ and $y \in W$ we have $xy \in E$ if and only if $xy \notin E'$. Also, given two coloured bipartite graphs $G_1 = (B_1, W_1, E_1)$ and $G_2 = (B_2, W_2, E_2)$, we denote by

- $G_1 \cup G_2$ the disjoint union of G_1 and G_2, i.e. $G_1 \cup G_2 = (B_1 \cup B_2, W_1 \cup W_2, E_1 \cup E_2)$.
- $G_1 \times G_2$ the bipartite join of G_1 and G_2, i.e. the bipartite complement of $\widetilde{G_1} \cup \widetilde{G_2}$. With a slight abuse of notation, when G_1 has only one vertex v, we will write $v \times G_2$ instead of $G_1 \times G_2$.

To each coloured class corresponds an uncoloured class that we obtain by simply forgetting the colouring of all the graphs.

2.2 Linear Clique-Width

The *linear clique-width* of a graph G, denoted $\text{lcw}(G)$, is the smallest number of labels needed to construct G by means of the following three operations:

- add a new vertex with label l.
- add all edges between vertices labeled l and all vertices labeled k, for $l \neq k$.
- relabel vertices labeled l to k.

A *linear clique-width expression* A for a graph G is an ordered sequence of these three operations that constructs G.

2.3 Classes of Graphs

The main object in this paper is the class of bi-complement reducible graphs that have been introduced in [7] and can be defined as follows.

Definition 1. *A* bi-complement reducible graph *(or* bi-cograph *for short) is a bipartite graph defined recursively as follows:*

(i) A graph on a single black or white vertex is a bi-cograph.
(ii) If G_1, G_2 are bi-cographs, then so is their disjoint union $G_1 \cup G_2$.
(iii) If G is a bi-cograph, then so is its bipartite complement $\widetilde{G}$.

It is not difficult to see that (iii) in the above definition could be replaced by:

(iii') If G_1, G_2 are bi-cographs, then so is their bipartite join $G_1 \times G_2$.

In [7], an induced subgraph characterisation for bi-cographs is also shown:

Proposition 1. *A bipartite graph is a bi-cograph if and only if it is (P_7, $Star_{123}$, Sun_4)-free (Fig. 1).*

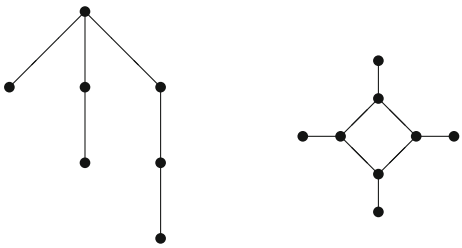


Fig. 1. Graphs Star_{123} (left) and Sun_4 (right)

In the present paper, we focus on two particular subclasses of bi-cographs. We introduce one of them under the name l-critical graphs in the next section. The second class consists of the bipartite complements of l-critical graphs.

3 l-Critical Graphs

The class of l-critical graphs is a subclass of bi-cographs and we define it in a similar way, i.e. inductively.

Definition 2. *A (coloured) l-*critical *graph is a bipartite graph defined recursively as follows:*

i. A graph on a single black or white vertex is l-critical.
ii. If G_1, G_2 are l-critical, then so is their union $G_1 \cup G_2$.
iii. If G is an l-critical graph, then the join of G with a single black vertex is an l-critical graph.

Remark 1. Note the asymmetry in this definition: we do not allow white dominating vertices while constructing an l-critical graph. However, once we have finished constructing it, we can forget the colouring, thus getting the uncoloured class of l-critical graphs. We will refer to both notions as simply l-critical graphs, but unless otherwise specified, we will be working with the coloured version.

Throughout the remainder of the paper, we denote by

– L the class of all l-critical graphs.

We now give a forbidden induced subgraph characterisation of the class L. We start with a characterisation of coloured graphs in L.

Lemma 1. *The following are equivalent for a coloured bipartite graph G:*

(a) $G \in L$;
(b) G contains no induced P_5 with white centre;
(c) any two black vertices of G have either comparable or disjoint neighbourhoods.

Proof. (a) $\Rightarrow$ (b): A P_5 with white centre is not in L, since it is not a disjoint union, and it does not have a black dominating vertex. Moreover, from the definition, L is hereditary, hence no graph in L contains a P_5 with white centre.
(b) $\Rightarrow$ (c): If two black vertices x and y have incomparable and non-disjoint neighbourhoods, then x, y together with a private neighbour of each and with a common neighbour induce a P_5 with white centre.
(c) $\Rightarrow$ (a): We want to show that, assuming (c), either G is disconnected, or it has a black dominating vertex (then use induction, and the fact that the condition (c) is hereditary). Suppose G is connected, and let b be a black vertex with a maximal (under set inclusion) neighbourhood. Let w be a white vertex, and consider a shortest path P from b to w (which exists, since G is connected). Write its vertices as $b = b_0, w_1, b_1, \ldots, b_{k-1}, w_k = w$ (where the vertices w_i are white, and the vertices b_i are black). If $k > 1$, then w_1 is a common neighbour to b and b_1, hence by (c) and maximality of the neighbourhood of b, $N(b_1) \subseteq N(b)$. In particular, w_2 and b are adjacent, and we have a shorter path between b and w, contradicting the choice of P. This shows $k = 1$, i.e. b and w are in fact adjacent, so b must be a dominating vertex. □

Forgetting the colours, we obtain the class of uncoloured l-critical graphs, which admits the following characterisation in terms of minimal forbidden induced subgraph.

Theorem 1. *A bipartite graph G is l-critical if and only if G is (P_6, C_6, domino, Sun_4)-free.*

Proof. The "only if" direction comes from the fact that any colouring of one of the four graphs in black and white contains a P_5 with white centre, hence by the previous lemma, none of the four graphs is l-critical.

Conversely, suppose G is (P_6, C_6, domino, Sun_4)-free. We show that there is colouring of G in black and white such that there is no P_5 with white centre. This is clear if G is P_5-free, so assume it is not. Without loss of generality, we can assume, in addition, that G is connected. Now find a P_5 induced by a, b, c, d

and e such that the neighbourhood of its middle vertex is maximal among all P_5's. Denote by S the part of G containing a, c and e, and by T the other part.

Let $x \in T$ be a neighbour of a. Then x is not a neighbour of e, otherwise a, b, c, d, e, x induce either a C_6 or a domino (depending on whether c and x are adjacent). Additionally, x must be a neighbour of c, otherwise the six vertices induce a P_6. With this in mind, let B be the set of neighbours of a and c, let D be the set of neighbours of e and c (in particular, $b \in B$ and $d \in D$), and let N_c be the set of neighbours of c that are not neighbours of a or e.

Suppose now that a vertex y in T is a non-neighbour of c (i.e. $y \notin B \cup N_c \cup D$, and y is also a non-neighbour of a and e). Find a path from y to c. Such a path must pass through $B \cup N_c \cup D = N(c)$. Let c' be the vertex of the path just before $N(c)$, and assume without loss of generality that y is adjacent to c'.

If c' has a neighbour b' in B, then c' must be adjacent to all vertices in D, since otherwise y, c', b', c, d', e form a P_6, where d' in a non-neighbour of c' in D. By symmetry, since c' is then adjacent to d, it must also be adjacent to all vertices in B. Moreover, c' is adjacent to all vertices in N_c, since if z is a non-neighbour of c' in N_c, then a, b, c, c', d, e, y, z induce a Sun_4.

Now if c' has a neighbour in $N_c \cup D$, then it must also be adjacent to b, otherwise we get another P_6, and like in the previous case, c' is adjacent to every vertex in $N(c)$. In any case, assuming c has a non-neighbour, we find a vertex c' with a strictly bigger neighbourhood than c (since y is adjacent to c' but not to c), such that a, b, c', d, e induce a P_5, contradicting our initial choice of P_5. In other words, S has a dominating vertex. If there was another P_5 with its centre in T, then that other P_5, together with a dominating vertex in S, would induce a domino. Hence if we colour S black and T white, we obtain a colouring of G with no P_5's with white centre. □

4 Unboundedness of Linear Clique-Width

The main result of this section is the unboundedness of linear clique-width in the classes of l-critical graphs and of their bipartite complements. To prove the result we will use an auxiliary graph parameter which provides a lower bound for linear clique-width.

Let G be a graph and A a linear clique-width expression for G. Clearly, A defines a linear order of the vertex set of G, i.e. a permutation π in the symmetric group $\mathcal{S}(V(G))$. Let us denote by $S_{\pi,i}$ the set consisting of the first i elements of the permutation, and by A_i the maximal prefix of A containing only the vertices of this set. If two vertices in $S_{\pi,i}$ have different neighbourhoods outside of the set, then they must have different labels in A_i, since otherwise in the rest of the expression we would not be able to add a neighbour to one of them without adding it to the other. Therefore, denoting by $\mu_{\pi,i}(G)$ the number of vertices in $S_{\pi,i}$ with pairwise different neighbourhoods outside of this set, we conclude that A uses at least

$$\mu_\pi(G) := \max_i \mu_{\pi,i}(G)$$

different labels to construct G. As a result, the linear clique-width is bounded from below by

$$\mu(G) := \min_{\pi \in \mathcal{S}(V(G))} \mu_\pi(G).$$

Therefore, to prove the main result of the section, it suffices to show that $\mu(G)$ is unbounded in the classes under consideration. In order to do that, we need a couple of technical lemmas describing the behaviour of $\mu(G)$ in some situations.

We introduce some notation for the coming part. Given a graph G and a linear order π of its vertices, we will write $v < w$ if v appears before w in the order, and $v < S$ if v appears before every vertex of a set S. Notice that the order on a graph induces an order on all of its subgraphs in the obvious way.

Every $i \in \{1, \ldots, n\}$ corresponds to a *cut in G with respect to π*, which separates the first i vertices in π from the rest of $V(G)$. It will be useful to mark cuts for which $\mu_{\pi,i}(G)$ is large. We will insert symbols $\alpha, \beta, \ldots$ into our ordered list of vertices to mark such cuts. If α marks a cut with $\mu_{\pi,i}(G) \geq t$, then a set of t vertices in $S_{\pi,i}$ with pairwise different neighbourhoods outside of $S_{\pi,i}$ will be called a *diversity witness of size t* for α. The largest t such that there exists a diversity witness of size t for α will be called the *diversity* of (the cut at) α.

Let H be a connected l-critical graph with $\mu(H) = t \geq 2$. Since H is connected and has at least two vertices, it contains both white and black vertices. Let $G = v \times (H \cup H \cup H)$ for a black vertex v, and label the vertices of the three copies of H by $A = \{a_i : 1 \leq i \leq n\}$, $B = \{b_i : 1 \leq i \leq n\}$, and $C = \{c_i : 1 \leq i \leq n\}$, respectively.

Lemma 2. $\mu(G) \geq t + 1$.

Proof. To prove the lemma, we fix an arbitrary permutation π of $V(G)$ and show that $\mu_\pi(G) \geq t + 1$. Let α, β, and γ be the three cuts of diversity of at least t in the three copies of H with respect to the restrictions of π into A, B, and C, respectively. Without loss of generality we assume that $\alpha \leq \beta \leq \gamma$ in π. Let $B' \subset B$ be a diversity witness of size t for β in B, i.e. $B' < \beta$, $|B'| = t$, and the vertices of B' have pairwise different neighbourhoods in the subset of B to the right of β.

Assume first that a vertex a of A appears after β. Since $\mu(H) \geq 2$, there exist vertices of A before α (and in particular before β). Therefore, since H is connected, there must be an edge $a_i a_j$ such that $a_i < \beta < a_j$. Since, by the definition of G, none of the vertices in B' is adjacent to a_j, set $B' \cup \{a_i\}$ is a diversity witness of size $t+1$ for β, i.e. $\mu_\pi(G) \geq t+1$. This conclusion allows us to assume, from now on, that

- $A < \beta$ and, by a similar argument, $\beta < C$ (we need $t \geq 2$ to make sure we do indeed have vertices of C after γ, and hence after β).

Suppose $v < \beta$. Since C has at least one white vertex, v has a neighbour in C and hence $B' \cup \{v\}$ is a diversity witness of size $t+1$ for β, i.e. $\mu_\pi(G) \geq t+1$. Therefore, in the rest of the proof we assume that

- $v > \beta$.

Assume B' contains a vertex b_i with no neighbour $b_j > \beta$ in B (observe that if such a vertex exists, then it is unique in B', since otherwise B' is not a diversity witness). If b_i is white, then for any black vertex $a_k \in A$, the set $B' \cup \{a_k\}$ is a diversity witness of size $t+1$ for β, because b_i is adjacent to v, while a_k is not (by the definition of G), and every vertex of B' different from b_i has a neighbour to the right of β, while a_k does not. Similarly, if b_i is black, then for any white vertex $a_k \in A$, the set $B' \cup \{a_k\}$ is a diversity witness of size $t+1$ for β in G, because b_i is not adjacent to v, while a_k is, and every vertex of B' different from b_i has a neighbour in B to the right of β, while a_k does not. In both cases, we have $\mu_\pi(G) \geq t+1$.

The above discussion allows us to assume that every vertex of B' has a neighbour in the subset of B to the right of β. Then for any vertex $a_k \in A$, the set $B' \cup \{a_k\}$ is a diversity witness of size $t+1$ for β in G, since a_k has no neighbours in B, i.e. $\mu_\pi(G) \geq t+1$. □

We recall that $\widetilde{G}$ denotes the bipartite complement of a bipartite graph G.

Lemma 3. *If G is a bipartite graph, then* $\mathrm{lcw}(\widetilde{G}) \leq 2 \cdot \mathrm{lcw}(G) + 2$.

Proof. We start with a linear clique-width expression A for G that uses $\mathrm{lcw}(G)$ labels, then modify it to obtain an expression A' using $2 \cdot \mathrm{lcw}(G)$ labels such that each label is reserved for only one part of the graph.

We now claim that we can modify A' to find a linear clique-width expression that uses $2 \cdot \mathrm{lcw}(G) + 2$ labels, in which the two additional labels i_b and i_w are reserved for inserting black and white vertices, respectively, and such that vertices are connected to all of their already constructed neighbours immediately as they are inserted. Indeed, say that a new vertex v is inserted in A' with label l. Whether an already constructed vertex w is a neighbour of v only depends on its label, so we can say that the set of already constructed neighbours of v is a union $\bigcup_{k \in \Lambda}\{w : w \text{ has label } k\}$, where Λ is a set of labels. In A', the label l might already be in use, so if we tried to connect v to all its already constructed neighbours right away, we might inadvertently add some extra edges (that do not appear in G) to the already constructed graph, between vertices labeled l and some other vertices. However, using a new, reserved label to insert v allows us to go around this. We can immediately connect it to all of its neighbours without changing the already constructed graph, and afterwards change the reserved label to the original label used for inserting v in A'. Proceeding inductively allows us to modify A' to an expression giving G with the desired properties.

A linear clique-width expression for $\widetilde{G}$ can be obtained from this modified expression by instead connecting newly inserted vertices to their non-neighbours in G of opposite colour that have already been inserted. □

Theorem 2. *Linear clique-width is unbounded in the class L of l-critical graphs and in the class $\widetilde{L}$ of their bipartite complements.*

Proof. Let $G_2 \simeq P_4$. It is easy to see that G_2 is a connected l-critical graph with $\mu(G_2) \geq 2$. Defining $G_k = v \times (G_{k-1} \cup G_{k-1} \cup G_{k-1})$ for $k > 2$, we conclude by

Lemma 2 that G_k is a connected l-critical graph with $\mu(G_k) \geq k$. Therefore, for each k, class L contains a graph of linear clique-width at least k. For class $\widetilde{L}$, a similar conclusion follows from Lemma 3. □

5 Minimality

The goal of this section is to show that the two classes of unbounded linear clique-width identified in the previous section are *minimal* hereditary classes where this parameter is unbounded. Moreover, we prove a more general result showing that the class of l-critical graphs and the class of their bipartite complements are the *only two minimal hereditary classes* of bi-cographs where the linear clique-width is unbounded.

To prove this result we will use a modification of linear clique-width defined as follows.

Definition 3. *The bipartite linear clique-width (or bi-linear clique-width, for short) of a bipartite graph G, denoted* $\mathrm{blcw}(G)$*, is the minimum number of labels necessary to construct G via a linear clique-width expression, but only allowing any given label to be used for either black or white vertices (we will call those labels black or white respectively).*

It is clear from the definition that $\mathrm{lcw}(G) \leq \mathrm{blcw}(G) \leq 2\mathrm{lcw}(G)$, so boundedness of one of the parameters is equivalent to boundedness of the other. We start with a couple of simple lemmas describing how this bi-linear clique-width behaves when taking bipartite unions or joins.

Lemma 4. *If $G_1, \ldots, G_r$ are bipartite graphs each with bi-linear clique-width at most k, then their bipartite union $\bigcup_{i=1}^{r} G_i$ and their bipartite join $\prod_{i=1}^{r} G_i$ have bi-linear clique-width at most $k+2$.*

Proof. Let $r = 2$. We construct G_1 using at most k labels in such a way that no vertices of different color ever receive the same label, then we relabel all black vertices to $k+1$ and all white vertices to $k+2$. We then construct G_2 using the first k labels (which are now unused). The bipartite union does not require any more operations. To construct the bipartite join, we connect vertices labeled by $k+1$ to all vertices labeled by white labels, except $k+2$, and we connect vertices labeled $k+2$ to all vertices labeled by black labels, except $k+1$. For general r, we continue by induction. □

Lemma 5. *If G_1 and G_2 are bipartite graphs with bi-linear clique-widths at most k and l respectively, then their union $G_1 \cup G_2$ and their join $G_1 \times G_2$ have bi-linear clique-width at most* $\max\{k+2, l\}$.

Proof. This lemma is a modification of the previous one in the case $r = 2$, and is proved in the same way. □

We are now ready to prove the main result of the section.

Theorem 3. *The class L of l-critical graphs and the class $\widetilde{L}$ of their bipartite complements are the only two minimal hereditary classes of bi-cographs of unbounded linear clique-width.*

Proof. To prove the theorem, we will show that for any graph $H \in L$ and any graph $K \in \widetilde{L}$, the class of (H, K)-free bi-cographs has bounded bi-linear clique-width. We proceed by induction on $|V(H)| + |V(K)|$. The base case $|V(H)| + |V(K)| = 2$ is trivial.

Assume now that the statement is true for $|V(H)| + |V(K)| \leq n$. More specifically, we assume that there exists a constant k such that by forbidding any graph $H \in L$ and any graph $K \in \widetilde{L}$ with $|V(H)| + |V(K)| \leq n$ we obtain a class of bi-cographs with bi-linear clique-width at most k. Now fix H and K with $|V(H)| + |V(K)| = n + 1$, and let G be an (H, K)-free bi-cograph with at least 2 vertices.

From the definition of l-critical graphs, and that of bi-cographs, we know:

- H either has a black dominating vertex, or is a bipartite disjoint union of strictly smaller graphs.
- K either has a black isolated vertex, or is a bipartite join of strictly smaller graphs.
- G is either a disjoint union or a join of strictly smaller graphs.

We distinguish between the following two cases.

1. H has a black dominating vertex v, or K has a black isolated vertex w. The two situations are analogous, so we only give details for the case where H has a black dominating vertex.
   (a) Suppose first that $G = \prod_{i=1}^{r} G_i$ is a bipartite join of smaller bi-cographs. If one of them, say G_1, contains an induced copy of $H - v$, then no other G_i has black vertices, since otherwise an induced copy of H arises, in which case G has the same bi-linear clique-width as G_1 (provided it is at least 3). Otherwise, each G_i is $(H - v)$-free and hence, by the inductive hypothesis, each G_i has bi-linear clique-width at most k, and so by Lemma 4, G itself has bi-linear clique-width at most $k + 2$.
   (b) Suppose now $G = \bigcup_{i=1}^{r} G_i$. Without loss of generality we can assume that each G_i is connected. Hence by the previous part, each G_i has bi-linear clique-width at most $k + 2$, and another application of Lemma 4 tells us G has bi-linear clique-width at most $k + 4$.
2. H is a disjoint union and K is a join. We represent H is a disjoint union $H^1 \cup H^2$ of two smaller graphs (chosen arbitrarily if the partition is not unique). Similarly, we represent K as a join $K^1 \times K^2$ of two smaller graphs. We will write $G = G_1 \star G_2$ to mean that G can be decomposed as either a union or a join of G_1 and G_2, which is always the case for a bi-cograph. If G is a union, then either G_1 is H_1-free or G_2 is H_2-free, and if G is a join, then

either G_1 is K^1-free or G_2 is K^2-free. In any case, by the inductive hypothesis, one of G_1 and G_2 has bi-linear clique-width at most k. Without loss of generality, assume $\text{blcw}(G_1) \leq k$. We write l_0 for the bi-linear clique-width of G, and l_1 for that of G_2. By Lemma 5, $l_0 \leq \max\{k+2, l_1\}$.

Note that G_2 can also be decomposed as $G_2 = G_1' \star G_2'$, and by the same argument as earlier, we can assume that $\text{blcw}(G_1') \leq k$. Denoting $l_2 := \text{blcw}(G_2')$, we have again $l_1 \leq \max\{k+2, l_2\}$ and applying the same argument repeatedly we obtain a sequence $l_i \leq \max\{k+2, l_{i+1}\}$. Note however that at each stage, l_i is the bi-linear clique-width of a smaller graph, and since this parameter cannot be larger the number of vertices of the graph, we eventually reach an $l_t \leq k+2$. But $l_i \leq k+2$ implies $l_{i-1} \leq \max\{k+2, l_i\} = k+2$, so that in fact all l_i, and in particular l_0, are bounded from above by $k+2$.

Putting everything together, we have $\text{blcw}(G) \leq k+4$, which finishes the proof. □

6 Conclusion

In this paper, we proved that in the class of bi-complement reducible graphs linear clique-width is unbounded and showed that this class contains exactly two minimal hereditary subclasses of unbounded linear clique-width. Our results lead naturally to a number of open problems.

It is known [14] that clique-width is bounded in the class of Star_{123}-free bipartite graphs. This class extends bi-cographs and hence contains both l-critical graphs and their bipartite complements. Are there other minimal hereditary subclasses of Star_{123}-free bipartite graphs with unbounded linear clique-width?

One more interesting class of bipartite graphs is known as chordal bipartite graphs [13]. It contains several subclasses, which are of primary importance in the context of clique-width. In particular, it contains bipartite permutation graphs, which is one of the first two hereditary classes that have been shown to be minimal of unbounded clique-width [12]. Recently, in [4], it was also shown that this is a minimal hereditary class of unbounded linear clique-width.

The class of chordal bipartite graphs also contains trees or, more generally, forests, where clique-width is known to be bounded but linear clique-width is not [1]. However, the problem of identifying minimal hereditary subclasses of forests of unbounded linear clique-width remains open.

Acknowledgment. Viktor Zamaraev acknowledges support from EPSRC, grant EP/P020372/1.

References

1. Adler, I., Kanté, M.M.: Linear rank-width and linear clique-width of trees. Theoret. Comput. Sci. **589**, 87–98 (2015)
2. Brandstädt, A., Dabrowski, K.K., Huang, S., Paulusma, D.: Bounding the clique-width of H-free split graphs. Discrete Appl. Math. **211**, 30–39 (2016)

3. Brignall, R., Korpelainen, N., Vatter, V.: Linear clique-width for hereditary classes of cographs. J. Graph Theory **84**, 501–511 (2017)
4. Collins, A., Foniok, J., Korpelainen, N., Lozin, V., Zamaraev, V.: Infinitely many minimal classes of graphs of unbounded clique-width. Discrete Appl. Math. (accepted). https://doi.org/10.1016/j.dam.2017.02.012
5. Dabrowski, K.K., Huang, S., Paulusma, D.: Classifying the clique-width of H-free bipartite graphs. Discrete Appl. Math. **200**, 43–51 (2016)
6. Fellows, M.R., Rosamond, F.A., Rotics, U., Szeider, S.: Clique-width is NP-complete. SIAM J. Discrete Math. **23**(2), 909–939 (2009)
7. Giakoumakis, V., Vanherpe, J.: Bi-complement reducible graphs. Adv. Appl. Math. **18**, 389–402 (1997)
8. Golumbic, M.C.: Trivially perfect graphs. Discrete Math. **24**, 105–107 (1978)
9. Gurski, F., Wanke, E.: On the relationship between NCL-width and linear NCL-width. Theoret. Comput. Sci. **347**, 76–89 (2005)
10. Heggernes, P., Meister, D., Papadopoulos, C.: Graphs of linear clique-width at most 3. Theoret. Comput. Sci. **412**, 5466–5486 (2011)
11. Heggernes, P., Meister, D., Papadopoulos, C.: Characterising the linear clique-width of a class of graphs by forbidden induced subgraphs. Discrete Appl. Math. **160**, 888–901 (2012)
12. Lozin, V.: Minimal classes of graphs of unbounded clique-width. Ann. Comb. **15**, 707–722 (2011)
13. Lozin, V., Rautenbach, D.: Chordal bipartite graphs of bounded tree- and clique-width. Discrete Math. **283**, 151–158 (2004)
14. Lozin, V.V., Volz, J.: The clique-width of bipartite graphs in monogenic classes. Int. J. Found. Comput. Sci. **19**, 477–494 (2008)
15. Yan, J.-H., Chen, J.-J., Chang, G.J.: Quasi-threshold graphs. Discrete Appl. Math. **69**, 247–255 (1996)

Linear Ramsey Numbers

Aistis Atminas[1], Vadim Lozin[2,3(✉)], and Viktor Zamaraev[4]

[1] Department of Mathematics, London School of Economics, Houghton Street, London WC2A 2AE, UK
A.Atminas@lse.ac.uk
[2] University of Warwick, Coventry, UK
V.Lozin@warwick.ac.uk
[3] Lobachevsky State University of Nizhniy Novgorod, Nizhny Novgorod, Russia
[4] Department of Computer Science, Durham University, South Road, Durham DH1 3LE, UK
viktor.zamaraev@gmail.com

Abstract. The Ramsey number $R_X(p,q)$ for a class of graphs X is the minimum n such that every graph in X with at least n vertices has either a clique of size p or an independent set of size q. We say that Ramsey number is *linear* in X if there is a constant k such that $R_X(p,q) \leq k(p+q)$ for all p,q. In the present paper we conjecture that Ramsey number is linear in X if and only if the co-chromatic number is bounded in X and determine Ramsey numbers for several classes of graphs that verify the conjecture.

1 Introduction

According to Ramsey's Theorem [8] for all natural p and q there exists a minimum number $R(p,q)$ such that every graph with at least $R(p,q)$ vertices has either a clique of size p or an independent set of size q.

The exact values of Ramsey numbers are known only for small values of p and q. However, with the restriction to specific classes of graphs, Ramsey numbers can be determined for all p and q. In particular, in [9] this problem was solved for planar graphs, while in [2] it was solved for line graphs, bipartite graphs, perfect graphs, P_4-free graphs and some other classes.

We denote the Ramsey number restricted to a class X by $R_X(p,q)$ and focus in the present paper on classes with a smallest speed of growth of $R_X(p,q)$. Clearly, $R_X(p,q)$ cannot be smaller than the minimum of p and q. We say that Ramsey number is *linear* in X if there is a constant k such that $R_X(p,q) \leq k(p+q)$ for all p,q.

It is not difficult to see that all classes of bounded co-chromatic number have linear Ramsey number, where the co-chromatic number of a graph G is the minimum k such that the vertex set of G can be partitioned into k subsets each of which is either a clique or an independent set. We conjecture that in the universe of hereditary classes of graphs the two notions coincide.

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 26–38, 2018.
https://doi.org/10.1007/978-3-319-94667-2_3

Conjecture 1. A hereditary graph class is of linear Ramsey number if and only if it is of bounded co-chromatic number.

A class of graphs is *hereditary* if it is closed under taking induced subgraphs. It is well known that a class of graphs is hereditary if and only if it can be characterized in terms of minimal forbidden induced subgraphs. Of particular interest in the present paper are *finitely defined* classes, i.e. classes defined by finitely many forbidden induced subgraphs.

In [4], it was conjectured that a finitely defined class X has bounded co-chromatic number if and only if the set of minimal forbidden induced subgraphs for X contains a P_3-free graph, the complement of a P_3-free graph, a forest (i.e. a graph without cycles) and the complement of a forest. Following this conjecture, we propose a restriction of our Conjecture 1 to the case of finitely defined classes as follows.

Conjecture 2. A finitely defined class X is of linear Ramsey number if and only if the set of minimal forbidden induced subgraphs for X contains a P_3-free graph, the complement of a P_3-free graph, a forest and the complement of a forest.

In Sect. 2, we prove the "only if" part of the conjecture and in Sect. 3 we determine Ramsey numbers for several classes of graphs that verify the "if" part of the conjecture. In the rest of the present section, we introduce basic terminology and notation.

All graphs in this paper are finite, undirected, without loops and multiple edges. The vertex set and the edge set of a graph G are denoted by $V(G)$ and $E(G)$, respectively. For a vertex $x \in V(G)$ we denote by $N(x)$ the neighbourhood of x, i.e. the set of vertices of G adjacent to x. The degree of x is $|N(x)|$. We say that x is *complete* to a subset $U \subset V(G)$ if $U \subseteq N(x)$ and *anticomplete* to U if $U \cap N(x) = \emptyset$. A subgraph of G induced by a subset of vertices $U \subseteq V(G)$ is denoted $G[U]$. By $\overline{G}$ we denote the complement of G and call it co-G.

A *clique* in a graph is a subset of pairwise adjacent vertices and an *independent set* is a subset of pairwise non-adjacent vertices. The size of a maximum clique in G is the *clique number* and the size of a maximum independent set is the *independence number* of G.

By K_n, C_n and P_n we denote a complete graph, a chordless cycle and a chordless path with n vertices, respectively. Also, $K_{n,m}$ is a complete bipartite graph with parts of size n and m, and $K_{1,n}$ is a star. A disjoint union of two graphs G and H is denoted $G + H$. In particular, pG is a disjoint union of p copies of G.

If a graph G does not contain induced subgraphs isomorphic to a graph H, then we say that G is H-free and call H a forbidden induced subgraph for G. In case of several forbidden induced subgraphs we list them in parentheses.

A *bipartite graph* is a graph whose vertices can be partitioned into two independent sets, and a *split graph* is a graph whose vertices can be partitioned into an independent set and a clique. A graph is bipartite if and only if it is free of odd cycles, and a graph is a split graph if and only if it is $(C_4, 2K_2, C_5)$-free [6].

2 Classes with Non-linear Ramsey Number

In this section, we prove the "only if" part of Conjecture 2.

Lemma 1. *For every fixed k, the class X_k of $(C_3, C_4, \ldots, C_k)$-free graphs is not of linear Ramsey number.*

Proof. Assume to the contrary that the Ramsey number for the class X_k is linear. Then there must exist a constant $t = t(k)$ such that any n-vertex graph from the class has an independent set of size at least n/t.

It is well-known (see e.g. [1]) that X_k contains n-vertex graphs with the independence number of order $O(n^{1-\epsilon}\ln(n))$, where $\epsilon > 0$ depends on k, which is smaller than n/t for large n. This contradiction shows that X_k is not of linear Ramsey number. □

Theorem 1. *Let X be a class of graphs defined by a finite set M of forbidden induced subgraph. If M does not contain a graph in at least one of the following four classes, then X is not of linear Ramsey number: P_3-free graphs, the complements of P_3-free graphs, forests, the complements of a forests.*

Proof. It is not difficult to see that a graph is P_3-free if and only if it is a disjoint union of cliques. The class of P_3-free graphs contains the graph $(q-1)K_{p-1}$ with $(q-1)(p-1)$ vertices with no clique of size p or independent set of size q, and hence this class is not of linear Ramsey number. Therefore, if M contains no P_3-free graph, then X_k contains all P_3-free graphs and hence is not of linear Ramsey number. Similarly, if M contains no $\overline{P}_3$-free graph, then X_k is not of linear Ramsey number.

Now assume that M contains no forest. Therefore, every graph in M contains a cycle. Since the number of graphs in M is finite, X contains the class of $(C_3, C_4, \ldots, C_k)$-free graphs for a finite value of k and hence is not of linear Ramsey number by Lemma 1. Applying the same arguments to the complements of graphs in X_k, we conclude that if M contains no co-forest, then X_k is not of linear Ramsey number. □

3 Classes with Linear Ramsey Number

In this section, we study classes of graphs defined by forbidden induced subgraphs with 4 vertices and determine Ramsey numbers for several classes in this family that verify the "if" part of Conjecture 2. All the eleven graphs on 4 vertices are represented in Fig. 1.

Below we list which of these graphs are P_3-free and which of them are forests (take the complements for $\overline{P}_3$-free graphs and for the complements of forests, respectively).

- P_3-free graphs: K_4, $\overline{K}_4$, $2K_2$, co-diamond, co-claw.
- Forests: $\overline{K}_4$, $2K_2$, P_4, co-diamond, co-paw, claw.

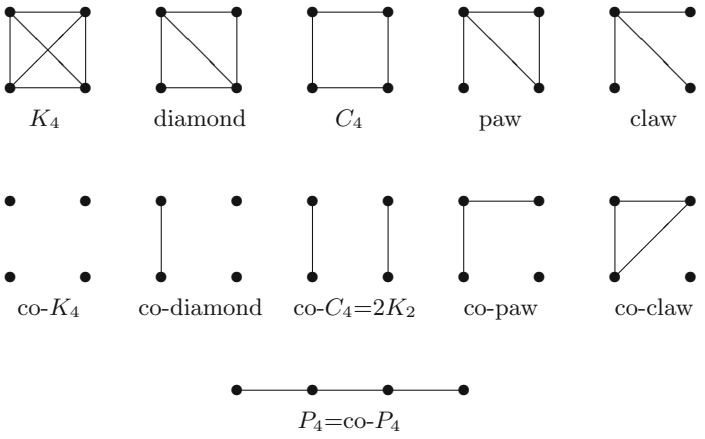


Fig. 1. All 4-vertex graphs

3.1 Claw- and Co-claw-free Graphs

Lemma 2. *If a (claw, co-claw)-free graph G contains a $\overline{K}_4$, then it is K_3-free.*

Proof. Assume G contains a $\overline{K}_4$ induced by $A = \{a_1, a_2, a_3, a_4\}$ and suppose by contradiction that G also contains a K_3 induced by $Z = \{x, y, z\}$.

Let first A be disjoint from Z. To avoid a co-claw, each vertex of A has a neighbour in Z and hence one of the vertices of Z is adjacent to two vertices of A, say x is adjacent to a_1 and a_2. Then, to avoid a claw, x has no other neighbours in A and y has a neighbour in $\{a_1, a_2\}$, say y is adjacent to a_1. This implies that y is adjacent to a_3 (else x, y, a_1, a_3 induce a co-claw) and similarly y is adjacent to a_4. But then y, a_1, a_2, a_3 induce a claw, a contradiction.

If A and Z are not disjoint, they have at most one vertex in common, say $a_4 = z$. Again, to avoid a co-claw, each vertex in $\{a_1, a_2, a_3\}$ has a neighbour in $\{x, y\}$ and hence, without loss of generality, x is adjacent to a_1 and a_2. But then x, a_1, a_2, a_4 induce a claw, a contradiction again. □

Lemma 3. *The maximum number of vertices in a (claw, co-claw, $K_4, \overline{K}_4$)-free graph is 9.*

Proof. Let G be a (claw, co-claw, $K_4, \overline{K}_4$)-free graph and let x be a vertex of G. Denote by A the set of neighbours and by B the set of non-neighbours of x. Clearly, A contains neither triangles nor anti-triangles, since otherwise either a K_4 or a claw arises. Therefore, A has at most 5 vertices, and similarly B has at most 5 vertices.

If $|A| = 5$, then $G[A]$ must be a C_5 induced by vertices, say, a_1, a_2, a_3, a_4, a_5 (listed along the cycle). In order to avoid a claw or K_4, each vertex of A can be adjacent to at most 2 vertices of B, which gives rise to at most 10 edges between A and B. On the other hand, to avoid a co-claw, each vertex of B must

be adjacent to at least 3 vertices of A. Therefore, B contains at most 3 vertices and hence $|V(G)| \leq 9$. Similarly, if $|B| = 5$, then $|V(G)| \leq 9$.

It remains to show that there exists a (claw, co-claw, $K_4, \overline{K}_4$)-free graph with 9 vertices. This graph can be constructed as follows. Start with a C_8 formed by the vertices $v_1, v_2, \ldots, v_8$. Then create a C_4 on the even-indexed vertices v_2, v_4, v_6, v_8 (listed along the cycle) and a $\overline{C}_4$ on the odd-indexed vertices v_1, v_3, v_5, v_7 (listed along the cycle in the complement). Finally, add one more vertex adjacent to the odd-indexed vertices. It is now a routine matter to check that the resulting graph is (claw, co-claw, $K_4, \overline{K}_4$)-free. This graph is known as the Paley graph of order $q = 3^2$. □

Theorem 2. *For the class A of (claw, co-claw)-free graphs and all $a, b \geq 3$,*

$$R_A(a, b) = \max(\lfloor (5a - 3)/2 \rfloor, \lfloor (5b - 3)/2 \rfloor),$$

unless $a = b = 4$ in which case $R_A(a, b) = 10$.

Proof. According to Lemma 2, the class of (claw, co-claw)-free graphs is the union of three classes:

- the class X of (claw, K_3)-free graphs,
- the class Y of (co-claw, $\overline{K}_3$)-free graphs and
- the class Z of (claw, co-claw, $K_4, \overline{K}_4$)-free graphs.

Clearly, $R_A(a, b) = \max(R_X(a, b), R_Y(a, b), R_Z(a, b))$.

Since K_3 is forbidden in X, we have $R_X(a, b) = R_X(3, b)$. Also, denoting by B the class of claw-free graphs, we conclude that $R_X(3, b) = R_B(3, b)$. As was shown in [2], $R_B(3, b) = \lfloor (5b - 3)/2) \rfloor$. Therefore, $R_X(a, b) = \lfloor (5b - 3)/2) \rfloor$. Similarly, $R_Y(a, b) = \lfloor (5a - 3)/2) \rfloor$.

In the class Z, for all $a, b \geq 4$ we have $R_Z(a, b) = 10$ by Lemma 3. Moreover, if additionally $\max(a, b) \geq 5$, then $R_Z(a, b) < \max(R_X(a, b), R_Y(a, b))$. For $a = b = 4$, we have $R_Z(4, 4) = 10 > 8 = \max(R_X(4, 4), R_Y(4, 4))$. Finally, it is not difficult to see that $R_Z(3, b) \leq R_X(3, b)$ and $R_Z(a, 3) \leq R_X(a, 3)$, and hence the result follows. □

3.2 Diamond- and Co-diamond-free Graphs

Lemma 4. *If a (diamond, co-diamond)-free graph G contains a $\overline{K}_4$, then it is bipartite.*

Proof. Assume G contains a $\overline{K}_4$. Let A be any maximal (with respect to inclusion) independent set containing the $\overline{K}_4$ and let $B = V(G) - A$. If B is empty, then G is edgeless (and hence bipartite). Suppose now B contains a vertex b. Then b has a neighbour a in A (else A is not maximal) and at most one non-neighbour (else a and b together with any two non-neighbours of b in A induce a co-diamond).

Assume B has two adjacent vertices, say b_1 and b_2. Since $|A| \geq 4$ and each of b_1 and b_2 has at most one non-neighbour in A, there are must be at least

two common neighbours of b_1 and b_2 in A, say a_1, a_2. But then a_1, a_2, b_1, b_2 induce a diamond. This contradiction shows that B is independent and hence G is bipartite. □

Lemma 5. *A co-diamond-free bipartite graph containing at least one edge is either a simplex (a bipartite graph in which every vertex has at most one non-neighbour in the opposite part) or a $K_{s,t} + K_1$ for some s and t.*

Proof. Assume $G = (A, B, E)$ is a co-diamond-free bipartite graph containing at least one edge. Then G cannot have two isolated vertices, since otherwise an edge together with two isolated vertices create an induced co-diamond.

Assume G has exactly one isolated vertex, say a, and let $G' = G - a$. Then any vertex $b \in V(G')$ is adjacent to every vertex in the opposite part of G'. Indeed, if b has a non-neighbour c in the opposite part, then a, b, c together with any neighbour of b (which exists because b is not isolated) induce a co-diamond. Therefore, G' is complete bipartite and hence $G = K_{s,t} + K_1$ for some s and t.

Finally, suppose G has no isolated vertices. Then every vertex $a \in A$ has at most one non-neighbour in B, since otherwise any two non-neighbours of a in B together with a and any neighbour of a (which exists because a is not isolated) induce a co-diamond. Similarly, every vertex $b \in B$ has at most one non-neighbour in A. Therefore, G is a simplex. □

Lemma 6. *The maximum number of vertices in a (diamond, co-diamond, $K_4, \overline{K}_4$)-free graph is 9.*

Proof. Let G be a (diamond, co-diamond, $K_4, \overline{K}_4$)-free graph and x be a vertex of G. Denote by A the set of neighbours and by B the set of non-neighbours of x. Then $G[A]$ is (P_3, K_3)-free, else G contains either a diamond or a K_4. Since $G[A]$ is P_3-free, every connected component of $G[A]$ is a clique and since this graph is K_3-free, every connected component has at most 2 vertices. If at least one of the components of $G[A]$ has 2 vertices, the number of components is at most 2 (since otherwise a co-diamond arises), in which case A has at most 4 vertices. If all the components of $G[A]$ have size 1, the number of components is at most 3 (since otherwise a $\overline{K}_4$ arises), in which case A has at most 3 vertices. Similarly, B has at most 4 vertices and hence $|V(G)| \leq 9$.

To conclude the proof, we observe that the Paley graph of order $q = 3^2$ described in the proof of Lemma 3 is (diamond, co-diamond, $K_4, \overline{K}_4$)-free. □

Theorem 3. *For the class A of (diamond, co-diamond)-free graphs and $a, b \geq 3$,*

$$R_A(a, b) = \max(2a - 1, 2b - 1),$$

unless $a, b \in \{4, 5\}$, in which case $R_A(a, b) = 10$, and unless $a = b = 3$, in which case $R_A(a, b) = 6$.

Proof. According to Lemma 4, in order to determine the value of $R_A(a, b)$, we analyze this number in three classes:

– the class X of co-diamond-free bipartite graphs,
– the class Y of the complements of graphs in X and
– the class Z of (diamond, co-diamond, $K_4, \overline{K}_4$)-free graphs.

In the class X of co-diamond-free bipartite graphs, $R_X(a,b) = 2b-1$, since every graph in this class with at least $2b-1$ contains an independent set of size b, while the graph $K_{b-1,b-1}$ contains neither an independent set of size b nor a clique of size $a \geq 3$. Similarly, $R_Y(a,b) = 2a-1$.

In the class Z of (diamond, co-diamond, $K_4, \overline{K}_4$)-free graphs, for all $a, b \geq 4$ we have $R_Z(a,b) = 10$ by Lemma 6. Moreover, if additionally $\max(a,b) \geq 6$, then $R_Z(a,b) < \max(R_X(a,b), R_Y(a,b))$. For $a, b \in \{4,5\}$, we have $R_Z(a,b) = 10 > \max(R_X(a,b), R_Y(a,b))$. Also, $R_Z(3,3) = 6$ (since $C_5 \in Z$) and hence $R_Z(3,3) > \max(R_X(3,3), R_Y(3,3))$. Finally, by direct inspection one can verify that Z contains no K_3-free graphs with more than 6 vertices and hence for $b \geq 4$ we have $R_Z(3,b) \leq R_X(3,b)$. Similarly, for $a \geq 4$ we have $R_Z(a,3) \leq R_Y(a,3)$. Thus for all values of $a, b \geq 3$, we have $R_A(a,b) = \max(2a-1, 2b-1)$, unless $a, b \in \{4,5\}$, in which case $R_A(a,b) = 10$, and unless $a = b = 3$, in which case $R_A(a,b) = 6$. □

3.3 $2K_2$- and C_4-Free Graphs

Theorem 4. *For the class A of $(2K_2, C_4)$-free graphs and all $a, b \geq 3$,*

$$R_A(a,b) = a + b.$$

Proof. Let G be a $(2K_2, C_4)$-free graph with $a+b$ vertices. If G is C_5-free, then it is a split graph and hence it contains either a clique of size a or an independent set of size b.

If G contains a C_5, then the remaining vertices of the graph can be partitioned into a clique U, whose vertices are complete to the cycle C_5, and an independent set W, whose vertices are anticomplete to the C_5 [3]. We have $|U|+|W| = a+b-5$ and hence either $|U| \geq a-2$ or $|W| \geq b-2$. In the first case, U together with any two adjacent vertices of the cycle C_5 create a clique of size a. In the second case, W together with any two non-adjacent vertices of the cycle create an independent set of size b. This shows that $R_A(a,b) \leq a+b$.

For the inverse inequality, we construct a graph G with $a+b-1$ vertices as follows: G consists of a cycle C_5, an independent set W of size $b-3$ anticomplete to the cycle and a clique U of size $a-3$ complete to both W and $V(C_5)$. It is not difficult to see that the size of a maximum clique in G is $a-1$ and the size of a maximum independent set in G is $b-1$. Therefore, $R_A(a,b) \geq a+b$. □

3.4 $2K_2$- and Diamond-Free Graphs

Lemma 7. *If a $(2K_2, \text{diamond})$-free graph G contains a K_4, then G is a split graph partitionable into a clique C and an independent set I such that every vertex of I has at most one neighbour in C.*

Proof. Let G be a ($2K_2$, diamond)-free graph containing a K_4. We extend the K_4 to any maximal (with respect to inclusion) clique and denote it by C. Also, denote $I = V(G) - C$.

Assume a vertex $a \in I$ has two neighbours b, c in C. It also has a non-neighbour d in C (else C is not maximal). But then a, b, c, d induce a diamond. This contradiction shows that any vertex of I has at most one neighbour in C.

Finally, assume two vertices $a, b \in I$ are adjacent. Since each of them has at most one neighbour in C and $|C| \geq 4$, there are two vertices $c, d \in C$ adjacent neither to a nor to b. But then a, b, c, d induce a $2K_2$. This contradiction shows that I is independent and completes the proof. □

Lemma 8. *Let G be a $(2K_2, diamond, K_4)$-free graph containing a K_3. Then G is 3-colorable.*

Proof. Denote a triangle K_3 in G by $T = \{a, b, c\}$, and for any subset $U \subseteq \{a, b, c\}$ let V_U be the subset of vertices outside of T such that $N(v) \cap T = U$ for each $v \in V_U$. Then

- $V_{a,b,c} = \emptyset$, since G is K_4-free.
- $V_{a,b} = V_{ac} = V_{bc} = \emptyset$, since G is diamond-free.
- $V_a, V_b, V_c, V_\emptyset$ are independent sets, since G is $2K_2$-free. For the same reason, every vertex of $V_\emptyset$ is isolated.

Then each of the following three sets $\{a\} \cup V_b$, $\{b\} \cup V_c$ and $\{c\} \cup V_a \cup V_\emptyset$ is independent and hence G is 3-colorable. □

The above two lemmas reduce the analysis to $(2K_2, K_3)$-free graphs. In order to characterize this class, let us say that G^* is an extended G (also known as a blow-up of G) if G^* is obtained from G by replacing the vertices of G with independent sets.

Lemma 9. *If G is a $(2K_2, K_3)$-free graph, then it is either bipartite or an extended $C_5 + K_1$.*

Proof. If G is C_5-free, then it is bipartite, because any cycle of length at least 7 contains an induced $2K_2$. Assume now that G contains a C_5 induced by a set $S = \{v_0, v_1, v_2, v_3, v_4\}$. To avoid an induced $2K_2$ or K_3, any vertex $u \notin S$ must be either anticomplete to S or have exactly two neighbours on the cycle of distance 2 from each other, i.e. $N(u) \cap S = \{v_i, v_{i+2}\}$ for some i (addition is taken modulo 5). Moreover, if $N(u) \cap S = \{v_i, v_{i+2}\}$ and $N(w) \cap S = \{v_j, v_{j+2}\}$, then

- if $i = j$ or $|i - j| > 1$, then u is not adjacent to w, since G is K_3-free.
- if $|i - j| = 1$, then u is adjacent to w, since G is $2K_2$-free.

Clearly, every vertex $u \notin S$, which is anticomplete to S, is isolated, and hence G is an extended $C_5 + K_1$. □

Theorem 5. *Let A be the class of $(2K_2, \text{diamond})$-free graphs. Then*

- *for $a = 3$, we have $R_A(a, b) = \lfloor 2.5(b-1) \rfloor + 1$,*
- *for $a = 4$, we have $R_A(a, b) = 3b - 2$,*
- *for $a \geq 5$, we have $R_A(a, b) = 3b - 2$ if $a < 2b$ and $R_A(a, b) = a + b - 1$ if $a \geq 2b$.*

Proof. As before, we split the analysis into several subclasses of A.

For the class X of $(2K_2$, diamond)-free graphs containing a K_4 and $a \geq 5$, we have $R_X(a, b) = a + b - 1$. Indeed, every split graph with $a + b - 1$ vertices contains either a clique of size a or an independent set of size b and hence $R_X(a, b) \leq a + b - 1$. On the other hand, the split graph with a clique C of size $a - 1$ and an independent set I of size $b - 1$ with a matching between C and I belongs to X and hence $R_X(a, b) \geq a + b - 1$.

For the class Y of 3-colorable $(2K_2$, diamond)-free graphs and for $a \geq 4$ we have $R_Y(a, b) = 3b - 2$. Indeed, a 3-colorable graph with $3b - 2$ vertices contains an independent set of size b and hence $R_Y \leq 3b - 2$. On the other hand, consider the graph G constructed from $b - 1$ triangles $T_i = \{a_i, b_i, c_i\}$ $(i = 1, 2, \ldots, b - 1)$ such that for all $j > i$,

- a_i is adjacent to b_j,
- b_i is adjacent to c_j,
- c_i is adjacent to a_j.

It is not difficult to see that G is 3-colorable $(2K_2$, diamond)-free graph with $3b - 3$ vertices containing neither a clique of size $a \geq 4$ nor an independent set of size b. Therefore, $R_Y \geq 3b - 2$.

For the class Z_0 of bipartite $2K_2$-free graphs, we have $R_{Z_0}(a, b) = 2b - 1$, which is easy to see. Finally, in the class Z_1 of graphs each of which is an extended $C_5 + K_1$, we have $R_{Z_1}(a, b) = \lfloor 2.5(b - 1) \rfloor + 1$. For an odd b, a maximum counterexample is constructed from a C_5 by replacing each vertex with an independent set of size $(b - 1)/2$. This graph has $\lfloor 2.5(b - 1) \rfloor$ vertices, the independence number $b - 1$ and the clique number $2 < a$. For an even b, a maximum counterexample is constructed from a C_5 by replacing two adjacent vertices of a C_5 with independent sets of size $b/2$ and the remaining vertices of the cycle with independent sets of size $b/2 - 1$. This again gives in total $\lfloor 2.5(b - 1) \rfloor$ vertices, and the independence number $b - 1$. Therefore, in the class $Z = Z_0 \cup Z_1$, we have $R_Z(a, b) = \max(R_{Z_0}(a, b), R_{Z_1}(a, b)) = \lfloor 2.5(b - 1) \rfloor + 1$.

Combining, we conclude that

- for $a = 3$, we have $R_A(a, b) = \lfloor 2.5(b - 1) \rfloor + 1$,
- for $a = 4$, we have $R_A(a, b) = 3b - 2$,
- for $a \geq 5$, we have $R_A(a, b) = 3b - 2$ if $a < 2b$ and $R_A(a, b) = a + b - 1$ if $a \geq 2b$.

□

3.5 The Class of (P_4, C_4, Co-claw)-Free Graphs

With start with a lemma characterizing the structure of graphs in this class, where we use the following well-known fact (see e.g. [5]): every P_4-free graph with at least two vertices is either disconnected or the complement to a disconnected graph.

Lemma 10. *Every disconnected (P_4, C_4, co-claw)-free graph is a collection of disjoint stars and every connected (P_4, C_4, co-claw)-free graph consists of a collection of disjoint stars plus a number of dominating vertices, i.e. vertices adjacent to all other vertices of the graph.*

Proof. Let G be a disconnected (P_4, C_4, co-claw)-free graph. Then every connected component of G is K_3-free, since a triangle in one of them together with a vertex from any other component create an induced co-claw. Therefore, every connected component of G is a bipartite graph. This graph is complete bipartite, because G is P_4-free, and it is a star, because G is C_4-free.

Now let G be a connected graph. Since G is P_4-free, $\overline{G}$ is disconnected. Let $C^1, \ldots, C^k$ ($k \geq 2$) be co-components of G, i.e. components in the complement of G. If at least two of them have more than 1 vertex, then an induced C_4 arises. Therefore, all co-components, except possibly one, have size 1, i.e. they are dominating vertices in G. If, say, C^1 is a co-component of size more than 1, then the subgraph of G induced by C^1 must be disconnected and hence it is a collection of stars. □

Theorem 6. *For the class A of (P_4, C_4, co-claw)-free graphs and all $a, b \geq 3$,*

$$R_A(a, b) = a + 2b - 4.$$

Proof. Let G be a graph in A with $a + 2b - 5$ vertices, $2b - 2$ of which induce a matching (a 1-regular graph with $b - 1$ edges) and the remaining $a - 3$ vertices are dominating in G. Then G has neither a clique of size a nor an independent set of size b. Therefore, $R_A(a, b) \geq a + 2b - 4$.

Conversely, let G be a graph in A with $a+2b-4$ vertices. If G is disconnected, then it is bipartite and hence at least one part in a bipartition of G has size at least b, i.e. G contains an independent set of size b. If G is connected, denote by C the set of dominating vertices in G. If $|C| \geq a - 1$, then either C itself (if $|C| \geq a$) or C together with a vertex not in C (if $|C| = a - 1$) create a clique of size a. So, let $|C| \leq a-2$. The graph $G-C$ is bipartite and has at least $2b-2$ vertices. If this graph has no independent set of size b, then in any bipartition of this graph each part contains exactly $b - 1$ vertices, and each vertex has a neighbour in the opposite part. But then $|C| = a - 2$ and therefore C together with any two adjacent vertices in $G - C$ create a clique of size a. □

3.6 The Class of (Co-diamond, Paw, Claw)-Free Graphs

Lemma 11. *Let G be a (co-diamond, paw, claw)-free graph.*

- *If G is connected, then it is either a path with at most 5 vertices or a cycle with at most 6 vertices or the complement of a graph of vertex degree at most 1.*
- *If G has two connected components, then either both components are complete graphs or one of the components is a single vertex and the other is the complement of a graph of vertex degree at most 1.*
- *If G has at least 3 connected components, then G is edgeless.*

Proof. Assume first that G is connected. It is known (see e.g. [7]) that every connected paw-free graphs is either K_3-free or complete multipartite. i.e. $\overline{P}_3$-free. If G is K_3-free, then together with the claw-freeness of G this implies that G has no vertices of degree more than 2, i.e. G is either a path or a cycle. To avoid an induced co-diamond, a path cannot have more than 5 vertices and a cycle cannot have more than 6 vertices. If G is complete multipartite, then each part has size at most 2, since otherwise an induced claw arises. In other words, the complement of G is a graph of vertex degree at most 1.

Assume now that G has two connected components. If each of them contains an edge, then both components are cliques, since otherwise two non-adjacent vertices in one of the components with two adjacent vertices in the other component create an induced co-diamond. If one of the components is a single vertex, then the other is $\overline{P}_3$-free (to avoid an induced co-diamond) and hence is the complement of a graph of vertex degree at most 1 (according to the previous paragraph).

Finally, let G have at least 3 connected components. If one of them contains an edge, then this edge together with two vertices from two other components form an induced co-diamond. Therefore, every component of G consists of a single vertex, i.e. G is edgeless. □

Theorem 7. *For the class A of (co-diamond, paw, claw)-free graphs and for all $a, b \geq 3$,*

$$R_A(a,3) = 2a - 1,$$
$$R_A(a,b) = \max(2a, b) \text{ for } b \geq 4,$$

except for the following four numbers $R_A(3,3) = 6$, $R_A(3,4) = R_A(3,5) = R_A(3,6) = 7$.

Proof. We start with the case $b = 3$. Since C_5 belongs to A, $R_A(3,3) = 6$, which covers the first of the four exceptional cases.

Let $a \geq 4$. The graph $2K_{a-1}$ with $2a-2$ vertices has neither cliques of size a nor independent sets of size 3, and hence $R_A(a,3) \geq 2a-1$. Conversely, let $G \in A$ be a graph with $2a-1 \geq 7$ vertices. If G is connected, then according to Lemma 11 G is the complement of a graph of vertex degree at most 1, and hence G has a clique of size a. If G has two connected components both of which are

cliques, then one of them has size at least a. If G has two connected components one of which is a single vertex, then either the second component has a couple of non-adjacent vertices, in which case an independent set of size 3 arises, or the second component is a clique of size more than a. If G has at least 3 connected components, then it contains an independent set of size more than 3. Therefore, $R_A(a,3) \leq 2a-1$ and hence $R_A(a,3) = 2a-1$ for $a \geq 4$.
From now on, $b \geq 4$. Consider the last three exceptional cases, i.e. let $a = 3$ and $4 \leq b \leq 6$.

The graph C_6 that belongs to our class has neither a clique of size 3 nor an independent set of size $b \geq 4$ and hence $R_A(a,b) \geq 7$ in these cases. Conversely, let $G \in A$ be a graph with at least 7 vertices. If G is connected, then it is the complement of a graph of vertex degree at most 1 and hence contains a clique of size 3. If G has two connected components each of which is a clique, then one of them has size at least 3. If G has two components one of which is a single vertex, then the other component has at least 6 vertices and also contains a clique of size 3. If G has at least 3 connected components, then G has an independent set of size $4 \leq b \leq 6$. Therefore, $R_A(a,b) = 7$ for $a = 3$ and $4 \leq b \leq 6$.

In the rest of the proof we assume that either $a \geq 4$ or $b \geq 7$. Denote $m = \max(2a, b)$. If $m = 2a$, then the graph $\overline{(a-1)K_2} + K_1$ with $2a-1$ vertices has neither cliques of size a nor independent sets of size $b \geq 7$. If $m = b$, then the edgeless graph with $b-1$ vertices has neither cliques of size a nor independent sets of size b. Therefore, $R_A(a,b) \geq m$.

Conversely, let G be a graph with at least $m \geq 7$ vertices. If G is connected, then it is the complement of a graph of vertex degree at most 1 and hence contains a clique of size a. If G has two connected components each of which is a clique, then one of them has size at least a. If G has two components one of which is a single vertex, then the other component has at least $2a-1$ vertices and also contains a clique of size a. If G has at least 3 connected components, then G has an independent set of size b. Therefore, $R_A(a,b) = m$. □

Acknowledgment. Vadim Lozin acknowledges support from the Russian Science Foundation Grant No. 17-11-01336.

References

1. Alon, N., Spencer, J.H.: The Probabilistic Method. Wiley, New York (2004)
2. Belmonte, R., Heggernes, P., van't Hof, P., Rafiey, A., Saei, R.: Graph classes and Ramsey numbers. Discrete Appl. Math. **173**, 16–27 (2014)
3. Blázsik, Z., Hujter, M., Pluhár, A., Tuza, Z.: Graphs with no induced C_4 and $2K_2$. Discrete Math. **115**, 51–55 (1993)
4. Chudnovsky, M., Seymour, P.: Extending Gyárfás-Sumner conjecture. J. Comb. Theory Ser. B **105**, 11–16 (2014)
5. Corneil, D.G., Lerchs, H., Stewart, B.L.: Complement reducible graphs. Discrete Appl. Math. **3**, 163–174 (1981)

6. Foldes, S., Hammer, P.L.: Split graphs. In: Congressus Numerantium, no. XIX, pp. 311–315 (1977)
7. Olariu, S.: Paw-free graphs. Inf. Process. Lett. **28**, 53–54 (1988)
8. Ramsey, F.P.: On a problem of formal logic. Proc. Lond. Math. Soc. **30**, 264–286 (1930)
9. Steinberg, R., Tovey, C.A.: Planar Ramsey numbers. J. Combin. Theory Ser. B **59**, 288–296 (1993)

Graphs that Are Not Pairwise Compatible: A New Proof Technique (Extended Abstract)

Pierluigi Baiocchi, Tiziana Calamoneri(✉), Angelo Monti, and Rossella Petreschi

Computer Science Department, "Sapienza" University of Rome, Rome, Italy
pierluigi.baiocchi@gmail.com, {calamo,monti,petreschi}@di.uniroma1.it

Abstract. A graph $G = (V, E)$ is a *pairwise compatibility graph* (PCG) if there exists an edge-weighted tree T and two non-negative real numbers d_{min} and d_{max}, $d_{min} \leq d_{max}$, such that each node $u \in V$ is uniquely associated to a leaf of T and there is an edge $(u, v) \in E$ if and only if $d_{min} \leq d_T(u, v) \leq d_{max}$, where $d_T(u, v)$ is the sum of the weights of the edges on the unique path $P_T(u, v)$ from u to v in T. Understanding which graph classes lie inside and which ones outside the PCG class is an important issue. Despite numerous efforts, a complete characterization of the PCG class is not known yet. In this paper we propose a new proof technique that allows us to show that some interesting classes of graphs have empty intersection with PCG. We demonstrate our technique by showing many graph classes that do not lie in PCG. As a side effect, we show a not pairwise compatibility planar graph with 8 nodes (i.e. C_8^2), so improving the previously known result concerning the smallest planar graph known not to be PCG.

Keywords: Phylogenetic tree reconstruction problem
Pairwise Compatibility Graphs (PCGs) · PCG recognition problem

1 Introduction

Graphs we deal with in this paper are motivated by a fundamental problem in computational biology, that is the reconstruction of ancestral relationships. It is known that the evolutionary history of a set of organisms is represented by a *phylogenetic tree*, i.e. a tree where leaves represent distinct known taxa while internal nodes are possible ancestors that might have led through evolution to this set of taxa. The edges of the tree are weighted in order to represent a kind of evolutionary distance among species. Given a set of taxa, the *phylogenetic tree reconstruction problem* consists in finding the "best" phylogenetic tree that

Partially supported by *Sapienza* University of Rome projects "Graph Algorithms for Phylogeny: a promising approach" and "Combinatorial structures and algorithms for problems in co-phylogeny".

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 39–51, 2018.
https://doi.org/10.1007/978-3-319-94667-2_4

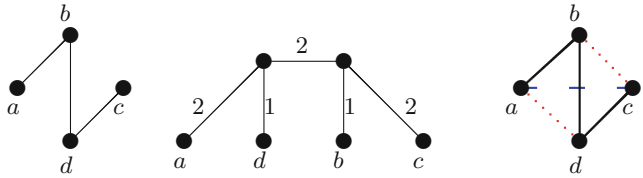
explains the given data. Since it is not completely clear what "best" means, the performance of the reconstruction algorithms is usually evaluated experimentally by comparing the tree produced by the algorithm with those partial subtrees that are unanimously recognized as "sure" by biologists. However, the tree reconstruction problem is proved to be NP-hard under many criteria of optimality, moreover real phylogenetic trees are usually huge, so testing these heuristics on real data is in general very difficult. This is the reason why it is common to exploit *sample techniques*, extracting relatively small subsets of taxa from large phylogenetic trees, according to some biologically-motivated constraints, and to test the reconstruction algorithms only on the smaller subtrees induced by the sample. The underlying idea is that the behavior of the algorithm on the whole tree will be more or less the same as on the sample. It has been observed that using, in the sample, very close or very distant taxa can create problems for phylogeny reconstruction algorithms [8] so, in selecting a sample from the leaves of the tree, the constraint of keeping the distance between any two leaves in the sample between two given positive integers d_{min} and d_{max} is used. This motivates the introduction of *pairwise compatibility graphs* (PCG). Indeed, given a phylogenetic tree T and integers d_{min}, d_{max}, we can associate a graph G, called the pairwise compatibility graph of T, whose nodes are the leaves of T and for which there is an edge between two nodes if the corresponding leaves in T are at a weighted distance within the interval $[d_{min}, d_{max}]$.

From a more theoretical point of view, we highlight that the problem of sampling a set of m leaves from a weighted tree T, such that their distance is within some interval $[d_{min}, d_{max}]$, reduces to selecting a clique of size m uniformly at random from the associated pairwise compatibility graph. As the sampling problem can be solved in polynomial time on PCGs [9], it follows that the *max clique problem* is solved in polynomial time on this class of graphs, if the edge-weighted tree T and the two values d_{min}, d_{max} are known or can be provided in polynomial time.

The previous reasonings motivate the interest of researchers in the so called *PCG recognition problem*, consisting in understanding whether, given a graph G, it is possible to determine an edge-weighted tree T and two integers d_{min}, d_{max} such that G is the associated pairwise compatibility graph; in this case G can be briefly denoted as $PCG(T, d_{min}, d_{max})$.

Figure 1(a) depicts a graph that is $PCG(T, 4, 5)$, where T is shown in Fig. 1(b). In general, T is not unique; here T is a *caterpillar*, i.e. a tree consisting of a central path, called spine, and nodes directly connected to that path. Due to their simple structure, caterpillars are the most used witness trees to show that a graph is PCG. However, it has been proven that there are some PCGs for which it is not possible to find a caterpillar as witness tree [4].

Due to the flexibility afforded in the construction of instances (i.e. choice of tree topology and values for d_{min} and d_{max}), when PCGs were introduced, it was also conjectured that all graphs are PCGs [9]. This conjecture has been refuted by proving the existence of some graphs not belonging to PCG. Namely, Yanhaona et al. [12] show a bipartite not PCG with 15 nodes. Mehnaz and Rahman [10]



a. b. c.

Fig. 1. a. A graph G. b. An edge-weighted caterpillar T such that $G = PCG(T, 4, 5)$. c. G where the PCG-coloring induced by triple $T, 4, 5$ is highlighted. (Color figure online)

generalize the technique in [12] to provide a class of bipartite graphs that are not PCGs. More recently, Durocher et al. [7] prove that there exists a not bipartite graph with 8 nodes that is not PCG; this is the smallest not planar graph that is not pairwise compatibility, since all graphs with at most 7 nodes are PCGs [4]. The authors of [7] provide also an example of a planar graph with 20 nodes that is not PCG. Moreover, it holds that, if a graph H is not a PCG, then every graph admitting H as an induced subgraph is also not a PCG [5]. Finally, a graph is not PCG if its complement has two 'far' induced subgraphs which are either a chordless cycle of at least four nodes or the complement of a cycle of length at least 5; two induced subgraphs are 'far' if they are both node disjoint and there is no edge connecting them [14].

From the other side, many graph classes have been proved to be in PCG, such as cliques and trees, cycles, single chord cycles, cacti, tree power graphs [12,13], interval graphs [2], triangle-free outerplanar 3-graphs [11] and Dilworth 2 graphs [6].

However, despite these results, the exact boundary of the PCG class remains unclear. In this paper, we move a concrete step in the direction of searching new graph classes that are not PCGs. To this aim, in Sect. 2 we introduce a new proof technique that allows us to show that some interesting classes have empty intersection with PCG. In particular, in Sect. 3 we show in detail the application of this technique on the class of graphs constructed as the square of a cycle. We prove that, for every $n \geq 8$, C_n^2 is not a PCG. Moreover, we show that by deleting any node from C_n^2 we get a PCG, thus proving that it does not contain any induced subgraph that is not PCG, i.e. we prove that C_n^2 is a *minimal* graph that is not PCG.

As a side effect, we prove that there exists also an 8 node planar graph that is not PCG, i.e. C_8^2, so improving the known result of a not pairwise compatibility planar graph with 20 nodes.

Finally, in Sect. 4, we present two other classes of graphs, obtained by modifying cycle graph in different ways, and we show that they are not PCGs through the application of our technique.

Due to the lack of space, for these latter classes, we only state the results referring the reader to [1] for the proofs' details.

2 Proof Technique

In this section, after introducing some definitions, we describe our proof technique, useful to prove that some classes of graphs have empty intersection with the class of PCGs, formally defined as follows.

Definition 1 [9]. *A graph $G = (V, E)$ is a* pairwise compatibility graph *(PCG) if there exists a tree T, a weight function assigning a positive real value to each edge of T, and two non-negative real numbers d_{min} and d_{max} with $d_{min} \leq d_{max}$, such that each node $u \in V$ is uniquely associated to a leaf of T and there is an edge $(u, v) \in E$ if and only if $d_{min} \leq d_T(u, v) \leq d_{max}$, where $d_T(u, v)$ is the sum of the weights of the edges on the unique path $P_T(u, v)$ from u to v in T.*

All trees in this paper are edge-weighted.

Given a graph $G = (V, E)$, we call *non-edges* of G the edges that do not belong to the graph. A *tri-coloring of G* is an edge labeling of the complete graph $K_{|V|}$ with labels from set {black, red, blue} such that all edges of $K_{|V|}$ that are in G are labeled black, while all the other edges of $K_{|V|}$ (i.e. the non-edges of G) are labeled either red or blue. A tri-coloring is called a *partial tri-coloring* if not all the non-edges of G are labeled.

Notice that, if $G = PCG(T, d_{min}, d_{max})$, some of its non-edges do not belong to G because the weights of the corresponding paths on T are strictly larger than d_{max}, while some other edges are not in G because the weights of the corresponding paths on T are strictly smaller than d_{min}. This motivates the following definition.

Definition 2. *Given a graph $G = PCG(T, d_{min}, d_{max})$, we call its* PCG-coloring *the tri-coloring $\mathcal{C}$ of G such that:*

- *(u, v) is* red *in $\mathcal{C}$ if $d_T(u, v) < d_{min}$,*
- *(u, v) is* black *in $\mathcal{C}$ if $d_{min} \leq d_T(u, v) \leq d_{max}$,*
- *(u, v) is* blue *in $\mathcal{C}$ if $d_T(u, v) > d_{max}$.*

In such a case, we will say that triple (T, d_{min}, d_{max}) induces *the PCG-coloring $\mathcal{C}$.*

In order to read the figures even in gray scale, we draw red edges as red-dotted and blue edges as blue-dashed in all the figures.

In Fig. 1(c) we highlight the PCG-coloring induced by the triple $(T, 4, 5)$ where T is the tree in Fig. 1(b).

The following definition formalizes that not all tri-colorings are PCG-colorings.

Definition 3. *A tri-coloring $\mathcal{C}$ (either partial or not) of a graph G is called a* forbidden PCG-coloring *if no triple (T, d_{min}, d_{max}) inducing $\mathcal{C}$ exists.*

Observe that a graph is PCG if and only if there exists a tri-coloring $\mathcal{C}$ that is a PCG-coloring for G. Besides, any induced subgraph H of a given $G = PCG(T, d_{min}, d_{max})$ is also PCG, indeed $H = PCG(T', d_{min}, d_{max})$, where T' is the subtree induced by the leaves corresponding to the nodes of H. Moreover, H inherits the PCG-coloring induced by triple (T, d_{min}, d_{max}) from G. Thus, if we were able to prove that H, although PCG, inherits a forbidden PCG-coloring from a tri-coloring $\mathcal{C}$ of G, then we would show that $\mathcal{C}$ cannot be a PCG-coloring for G in any way. This is the core of our proof technique:

Given a graph G that we want to prove not to be PCG:

1. *list some forbidden PCG-colorings of particular graphs that are induced subgraphs of G;*
2. *show that each tri-coloring of G induces a forbidden PCG-coloring in at least an induced subgraph;*
3. *conclude that G is not PCG, since all its tri-colorings are proved to be forbidden.*

3 The Square of a Cycle

In this section we exploit the proof technique just described on a particular class of graphs, i.e. the square of a cycle; we recall that the square G^2 of a graph G is a new graph whose node set coincides with the node set of G, and an edge (u, v) is in G^2 if either (u, v) is in G or (u, w) and (w, v) are both in G for some node w.

3.1 Forbidden Tri-Colorings of Some Subgraphs of C_n^2

In agreement with the proof technique described in Sect. 2, as a first step, here we highlight forbidden partial tri-colorings of paths P_n, $n \geq 3$ and cycles C_n, $n \geq 4$. Moreover, we prove forbidden colorings and partial forbidden colorings (for short **f-c**) for some graphs that are induced subgraphs of C_n^2 (see Figs. 2 and 3).

Given a graph $G = (V, E)$ and a subset $S \subseteq V$, we denote by $G[S]$ the subgraph of G induced by nodes in S.

A *subtree induced by a set of leaves* of T is the minimal subtree of T which contains those leaves. In particular, we denote by T_{uvw} the subtree of a tree induced by three leaves u, v and w.

The following lemma from [12] will be largely used:

Lemma 1. *Let T be a tree, and u, v and w be three leaves of T such that $d_T(u, v) \geq \max\{d_T(u, w), d_T(v, w)\}$. Let x be a leaf of T other than u, v, w. Then, $d_T(w, x) \leq \max\{d_T(u, x), d_T(v, x)\}$.*

It is known that P_m is a PCG [13]; the following lemma gives some constraints to the associated PCG-coloring.

Lemma 2. *Let P_m, $m \geq 4$, be path $v_1, \ldots, v_m$ and let $\mathcal{C}$ be one of its PCG-colorings. If all non-edges (v_1, v_i), $3 \leq i \leq m-1$, and (v_2, v_m) are colored with blue in $\mathcal{C}$, then also non-edge (v_1, v_m) is colored with blue in $\mathcal{C}$.*

Proof. Let $\mathcal{C}$ be the PCG-coloring of P_m induced by triple (T, d_{min}, d_{max}). We apply Lemma 1 iteratively.

First consider nodes v_1, v_2, v_3 and v_4 as u, w, v and x: $P_T(v_1, v_3)$ is easily the largest path in $T_{v_1 v_3 v_2}$; then $d_T(v_2, v_4) \leq \max\{d_T(v_1, v_4), d_T(v_3, v_4)\} = d_T(v_1, v_4)$. This is because (v_1, v_4) is a blue non-edge by hypothesis while (v_3, v_4) is an edge.

Now repeat the reasoning with nodes v_1, v_2, v_i and v_{i+1}, $4 \leq i < m$, as u, w, v and x, exploiting that at the previous step we have obtained that $d_T(v_2, v_i) \leq d_T(v_1, v_i)$: in $T_{v_1 v_i v_2}$, $P_T(v_1, v_i)$ is the largest path and so $d_T(v_2, v_{i+1}) \leq \max\{d_T(v_1, v_{i+1}), d_T(v_i, v_{i+1}) = d_T(v_1, v_{i+1})$ since (v_1, v_{i+1}) is a blue non-edge while (v_i, v_{i+1}) is an edge.

Posing $i = m-1$, we get that $d_T(v_2, v_m) \leq d_T(v_1, v_m)$; since non-edge (v_2, v_m) is blue by hypothesis, (v_1, v_m) is blue, too. □

Given a graph, in order to ease the exposition, we call *2-non-edge* a non-edge between nodes that are at distance 2 in the graph.

Lemma 3. *Let P_n, $n \geq 3$, be a path. Any PCG-coloring of P_n that has at least one red non-edge but no red 2-non-edges is forbidden.*

Proof. If $n = 3$, there is a unique non-edge and it is a 2-non-edge; so, the claim trivially follows.

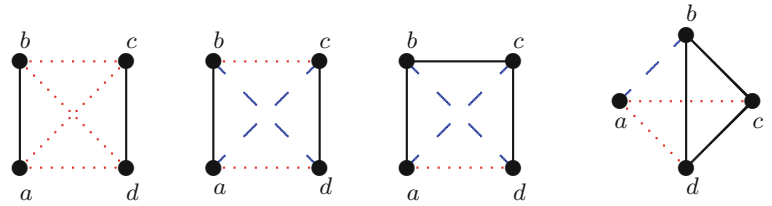
So, let it be $n \geq 4$ and consider a triple (T, d_{min}, d_{max}) inducing a PCG-coloring with at least a red non-edge. Among all red non-edges, let (v_i, v_j) – $i < j$– be the one such that $j - i$ is minimum. Assume by contradiction, $j - i > 2$. Consider now the subpath P' induced by $v_i, \ldots, v_j$. P' has at least 4 nodes and inherits the PCG-coloring from P_n; in it, there is only a red non-edge (i.e. the non-edge connecting v_i and v_j). P' satisfies the hypothesis of Lemma 2, hence (v_i, v_j) must be blue, against the hypothesis that it is red. □

Lemma 4. *Let C_n, $n \geq 4$, be a cycle. Then any PCG-coloring of C_n that does not have red 2-non-edges is forbidden.*

Proof. Let $C_n = PCG(T, d_{min}, d_{max})$, $n \geq 4$; from [13], at least a non-edge (u, v) such that $d_T(u, v) < d_{min}$. In our setting, this means that every PCG-coloring of C_n, $n \geq 4$, has at least one red non-edge. By contradiction, w.l.o.g. assume that this non-edge is (v_1, v_i), with $4 \leq i < n-1$. We apply Lemma 3 on the induced P_i and the claim follows. □

Lemma 5. *The tri-colorings in Fig. 2 are forbidden PCG-colorings.*

Proof. We prove separately that each tri-coloring is forbidden.



a. **f-c**$(2K_2)a$ b. **f-c**$(2K_2)b$ c. **f-c**(P_4) d. **f-c**$(K_3 \cup K_1)$

Fig. 2. Forbidden tri-colorings of some graphs. (Color figure online)

Forbidden Tri-Coloring f-c$(2K_2)a$:

We obtain that the tri-coloring in Fig. 2(a) is forbidden by rephrasing Lemma 6 of [7] with our nomenclature.

The other proofs are all by contradiction and proceed as follows: for each tri-coloring in Fig. 2, we assume that it is a feasible PCG-coloring induced by a triple (T, d_{min}, d_{max}) and show that this assumption contradicts Lemma 1.

Forbidden Tri-Coloring f-c$(2K_2)b$:

From the tri-coloring in Fig. 2(b) we have that

$$d_T(b,c) < d_{min} \leq d_T(a,b) \leq d_{max} < d_T(a,c).$$

Thus $P_T(a,c)$ is the largest path in $T_{a,b,c}$. By Lemma 1, for leaf d it must be: $d_T(b,d) \leq \max\{d_T(a,d), d_T(c,d)\} = d_T(c,d)$ while from the tri-coloring it holds that $d_T(c,d) \leq d_{max} < d_T(b,d)$, a contradiction.

Forbidden Tri-Coloring f-c(P_4):

From the tri-coloring in Fig. 2(c) we have that

$$d_T(a,b), d_T(b,c) \leq d_{max} < d_T(a,c).$$

Thus $P_T(a,c)$ is the largest path in $T_{a,b,c}$. By Lemma 1, for leaf d we have: $d_T(b,d) \leq \max\{d_T(a,d), d_T(c,d)\} = d_T(c,d)$ while from the tri-coloring it holds that $d_T(c,d) \leq d_{max} < d_T(b,d)$, a contradiction.

Forbidden Tri-Coloring f-c$(K_3 \cup K_1)$:

From the tri-coloring in Fig. 2(d) we have that

$$d_T(a,d), d_T(a,c) < d_{min} \leq d_T(c,d).$$

Thus $P_T(c,d)$ is the largest path in $T_{a,c,d}$. By Lemma 1, for leaf b it must be: $d_T(a,b) \leq \max\{d_T(c,b), d_T(d,b)\}$ while from the tri-coloring it holds that $d_T(c,b), d_T(d,b) < d_{max} \leq d_T(a,b)$, a contradiction. □

Lemma 6. *The partial tri-coloring in Fig. 3 is a forbidden PCG-coloring.*

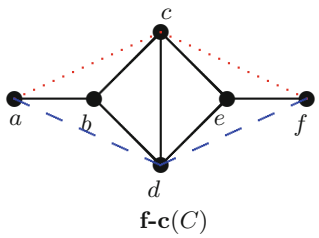


Fig. 3. A forbidden coloring of a graph that is an induced subgraph of C_n^2, with $n \geq 10$. (Color figure online)

Proof. Using the result of Lemma 5, we again prove that each tri-coloring is forbidden by contradiction.

From the tri-coloring in Fig. 3, extract the inherited PCG-colorings for the two subgraphs $G[a, c, d, e]$ and $G[b, c, d, f]$. To avoid **f-c**$(K_3 \cup K_1)$, the non-edges (a, e) and (b, f) are both blue. Now we distinguish the two possible cases for the color of the non-edge (a, f):

(a, f) **is a red non-edge:** consider the PCG-coloring for subgraph $G[a, b, e, f]$. To avoid **f-c**$(2K_2)b$, non-edge (b, e) has to be blue. This implies that the PCG-coloring for path $G[a, b, d, e, f]$ has all the 2-non-edges with color blue while the non-edge (a, f) is red. This is in contradiction with Lemma 3.

(a, f) **is a blue non-edge:** in this case consider Lemma 1 applied to tree $T_{a,d,f}$. We distinguish the three cases for the largest path among $P_T(a, d)$, $P_T(a, f)$ and $P_T(d, f)$:

the largest path is $P_T(a, d)$: for leaf b it must be:

$$d_T(f, b) \leq \max\{d_T(a, b), d_T(d, b)\}$$

while from the tri-coloring $d_T(a, b), d_T(d, b) \leq d_{max} < d_T(f, b)$.

the largest path is $P_T(a, f)$: for leaf c it must be:

$$d_T(d, c) \leq \max\{d_T(a, c), d_T(f, c)\}$$

while from the tri-coloring $d_T(a, c), d_T(f, c) < d_{min} \leq d_T(d, c)$.

the largest path is $P_T(d, f)$: for leaf e it must be:

$$d_T(a, c) \leq \max\{d_T(d, c), d_T(f, c)\}$$

while from the tri-coloring $d_T(d, c), d_T(f, c) \leq d_{max} < d_T(a, c)$.

In all the three cases, a contradiction arises. □

3.2 Graph C_n^2, $n \geq 8$, Is Not PCG

We recall that all graphs with at most 7 nodes are PCG [4] and that cycles are PCGs [12], so we focus on $n \geq 8$.

For easing the proofs, the nodes of C_n^2 will be indexed with values in the finite group $\mathbb{Z}_n$ of the integers modulo n, i.e. $V(C_n^2) = \{v_0, v_1, \ldots, v_{n-1}\}$. As a consequence, for each pair v_i, v_j, the edge (v_i, v_j) belongs to C_n^2 if and only if $j - i \in \{1, 2, n-1, n-2\}$.

Before proving that C_n^2 is not PCG, we need some *ad-hoc* forbidden PCG-colorings for C_n^2. Due to the lack of space, we omit the proof, that can be found in [1].

Given a PCG-coloring of C_n^2, we call *red-node* a node v of C_n^2 if all the non-edges incident on v are of red color.

Lemma 7. *Let C_n^2, $n \geq 8$, be a square cycle. Then:*

1. *Any PCG-coloring of C_n^2 where all the 2-non-edges are blue is forbidden.*
2. *Any PCG-coloring of C_n^2 having two red non-edges from a common non red-node to two adjacent nodes is forbidden.*
3. *Any PCG-coloring of C_n^2 having two adjacent red-nodes is forbidden.*

Now we show other two *ad-hoc* forbidden PCG-colorings that hold only for $n \geq 10$ because in the proof we exploit $\mathbf{f} - \mathbf{c}(C)$. Hence the two cases $n = 8$ and $n = 9$ have to be handled separately. Due to the lack of space, their proof are omitted in this extended abstract and can be found in [1].

Lemma 8. *Let C_n^2, $n \geq 10$, be a square cycle. Then:*

1. *Any PCG-coloring of C_n^2 with a triple of nodes (v_i, v_{i+4}, v_{i+8}), $0 \leq i < n$, such that v_{i+8} is the only non red-node (in this triple) is forbidden.*
2. *Any PCG-coloring of C_n^2 with a triple of nodes (v_{i-6}, v_{i-3}, v_i), $0 \leq i < n$, such that v_{i-6} is the only non red-node (in this triple) is forbidden.*

We are now ready to prove that C_n^2 is not PCG.

Theorem 1. *Graph C_n^2, $n \geq 10$, is not a PCG.*

Proof. The proof is by contradiction. Let (v_i, v_{i+4}) be a red 2-non-edge in C_n^2 (such a non-edge must exist by Lemma 7.1). Consider now the induced path $G[v_i, v_{i+1}, v_{i+3}, v_{i+4}]$. In this path we have the red non-edge (v_i, v_{i+4}) thus, due to $\mathbf{f} - \mathbf{c}(P_4)$, one of the non-edges (v_i, v_{i+3}) and (v_{i+1}, v_{i+4}) is red, too and at least one of the nodes v_i and v_{i+4} is the end-point of two red non-edges toward adjacent nodes. Hence one of these nodes is a red-node (see Lemma 7.2). Reindexing the nodes of C_n^2, this red-node is node v_0.

Consider now the induced subgraph $G[v_{n-3}, v_{n-1}, v_0, v_1, v_2, v_4]$. In this subgraph the non-edges (v_{n-3}, v_0) and (v_0, v_4) are red and, due to $\mathbf{f} - \mathbf{c}(C)$, at least one of the non-edges (v_{n-3}, v_1) and (v_1, v_4) is red. We consider two cases:

non-edge (v_1, v_4) is red

The two non-edges (v_0, v_4) and (v_1, v_4) are red so, by Lemma 7.2, node v_4 is a red-node. Considering the triple of nodes (v_0, v_4, v_8), by Lemma 8.1, node v_8 is a red-node, too. We can iterate this reasoning on the triple (v_4, v_8, v_{12}) and so on, and finally obtaining that $V^* = \{v_i \mid i \equiv 0 \pmod 4\}$ is a set of red-nodes in the PCG-coloring. Moreover each node v_i, with $i \not\equiv 0 \pmod 4$, is adjacent to some node in V^* thus, by Lemma 7.3, n is a multiple of 4 (and $n \geq 12$) and set V^* contains all the red-nodes of the PCG-coloring. Consider now the cycle induced by all the nodes having an odd index, i.e. $G[v_1, v_3, v_5, \ldots, v_{n-1}]$. This cycle is $\frac{n}{2} \geq 6$ long thus, by Lemma 4, it contains at least a red non-edge. Let (v_i, v_j) be one of these red non-edges. Node v_j is necessarily adjacent to a node in V^*, hence there are two red non-edges from adjacent nodes incident toward v_i in C_n^2 implying that v_i is a red-node (by Lemma 7.2). This contradicts the fact that $v_i \notin V^*$.

non-edge (v_{n-3}, v_1) is red

The proof is analogous to the previous one: due to the two red non-edges (v_{n-3}, v_0) and (v_{n-3}, v_1), by Lemma 7.2, node v_{n-3} is a red-node. Considering the triple of nodes (v_{n-6}, v_{n-3}, v_0) in Lemma 8.2, node v_{n-6} is a red-node, too. We can iterate this reasoning on the triple $(v_{n-9}, v_{n-6}, v_{n-3})$ and so on finally obtaining that $V^* = \{v_i \mid i \equiv 0 \pmod 3\}$ is a set of red-nodes in the PCG-coloring. Moreover, each node v_i, with $i \not\equiv 0 \pmod 3$, is adjacent to some node in V^* so, due to Lemma 7.3, n is a multiple of 3 (and $n \geq 12$) and set V^* contains all the red-nodes of the PCG-coloring. Consider now the cycle induced by all the nodes that are not in V^*, i.e. $G[1, 2, 4, \ldots, n-2, n-1]$. This cycle has length at least 8 and, by Lemma 4, there is at least a red non-edge connecting two nodes of the cycle. Let (v_i, v_j) be one of these red non-edges. Node v_j is adjacent to a node in V^*, so v_i is the end-point of two red non-edges toward adjacent nodes in C_n^2 as a consequence v_i is a red-node (by Lemma 7.2). This contradicts the fact that $v_i \notin V^*$.

Theorem 2. *Graph C_8^2 is not a PCG.*

Corollary 1. *Graph C_8^2 is the smallest planar graph that is not PCG.*

Theorem 3. *Graph C_9^2 is not a PCG.*

3.3 Graph $C_n^2, n \geq 8$, Is a Minimal Graph that Is Not PCG

Recall that if a graph contains as induced subgraph a not PCG, then it is not PCG, too. We call *minimal not PCG* a graph that is not PCG and it does not contain any induced proper subgraph that is not PCG. (It is worth to be noted that PCG is closed under taking induced subgraphs.)

In this subsection we prove that C_n^2 is a minimal not PCG. The proof is constructive and it provides an edge-weighted tree T and two values d_{min} and d_{max} such that $PCG(T, d_{min}, d_{max}) = C_n^2 \setminus \{x\}$ for any node x of C_n^2.

Theorem 4. *C_n^2, $n \geq 8$, is a minimal not PCG.*

Proof. Consider the graph C_n^2, $n \geq 8$. To prove the theorem we remove from the graph a node x and prove that the new graph G' is PCG. Without loss of generality assume that $x = v_n$. We construct a tree T such that $G' = \text{PCG}(T, 2n-2, 2n+4)$. We consider the following two cases depending on whether n is an even or an odd number.

- *n is an odd number.* Tree T is a caterpillar with $n-1$ internal nodes we denote as $x_1, x_2, \ldots, x_{\frac{n-1}{2}-1}, y, x_{\frac{n-1}{2}}, \ldots, x_{n-2}$. The internal nodes induce a path from x_1 to x_{n-2} and edges (x_i, x_{i+1}), $1 \leq i < (n-1)/2 - 1$ and $(n-1)/2 \leq i < n-2$, have weight 2. Edges $(x_{\frac{n-1}{2}-1}, y)$ and $(y, x_{\frac{n-1}{2}})$ have weight 1. Leaves v_i, $1 \leq i \leq n-2$, are connected to x_i with edges of weight n. Finally leaf v_{n-1} is connected to the node y with an edge of weight 3. See Fig. 4.a
- *n is an even number.* Tree T is a caterpillar with $n-1$ internal nodes we denote as $x_1, x_2, \ldots, x_{n-1}$. The internal nodes $x_1, \ldots, x_{n-1}$ induce a path and edges (x_i, x_{i+1}), $1 \leq i < n-1$, have weight 2.

Leaves v_i, $1 \leq i < n$, are connected to x_i with edges of weight n. Finally v_{n-1} is connected to $x_{\frac{n-2}{2}}$ with an edge of weight 3. See Fig. 4(b). □

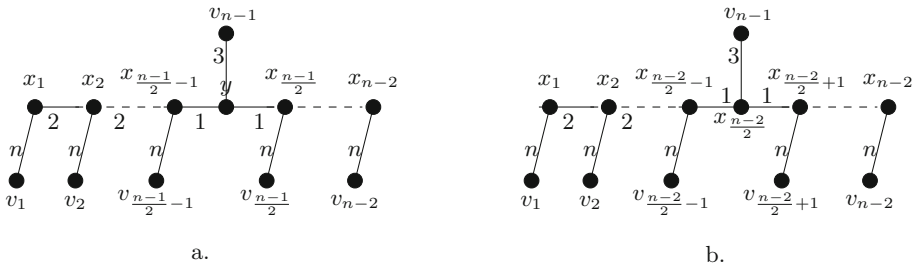


Fig. 4. Caterpillars for the proof of Theorem 4: a. n odd; b. n even.

4 Other Results Due to the Application of Our Technique

In this section we get two further results applying again the technique introduced in Sect. 2. Due to the lack of space, we omit the proofs, that can although be found in [1]. The graph classes we consider are obtained by operating in different ways on cycles and are very interesting in this context because both are connected to some open problems.

4.1 The Wheel

Wheels W_{n+1} are n length cycles C_n whose nodes are all connected with a universal node.

Wheel W_{6+1} is PCG and it is the only graph with 7 nodes whose witness tree is not a caterpillar [4]. Moreover, it has been proven in [3] that also the larger wheels up to W_{10+1} do not have a caterpillar as a witness tree but, up to now, no other witness trees are known for these graphs and, in general, it has been left open to understand whether wheels with at least 8 nodes are PCGs or not.

Using our technique, we prove the following theorem.

Theorem 5. *Wheel W_{7+1} is a PCG while wheels W_{n+1}, $n \geq 8$, are minimal not PCGs.*

4.2 The Strong Product of a Cycle and P_2

Given two graphs G and H, their *strong product* $G \square H$ is a graph whose node set is the Cartesian product of the node sets of the two graphs, and there is an edge between nodes (u, v) and (u', v') if and only if either $u = u'$ and (v, v') is an edge of H or $v = v'$ and (u, u') is an edge of G.

We recall that $C_4 \square P_2$ has already been proved not to be PCG [7] but nothing is known for $n > 4$. Recalling that all graphs with 7 nodes or less are PCGs, our result is the following.

Theorem 6. *The graphs obtained as strong product $C_n \square P_2$, $n \geq 4$ are minimal not PCGs.*

5 Conclusions

In this paper we proposed a new proof technique to show that graphs are not PCGs. As an example, we applied it to the square of cycles, to wheels and to $C_n \square P_2$. As a side effect, we show that the smallest planar graph not to be PCG has not 20 nodes, as previously known, but only 8.

Even if all these classes are obtained by operating on cycles, we think that this technique can be potentially used to position outside PCG many other graph classes no related to cycles. This represents an important step toward the solution of the very general open problem consisting in demarcating the boundary of the PCG class.

References

1. Baiocchi, P., Calamoneri, T., Monti, A., Petreschi, R.: Some classes of graphs that are not PCGs. arXiv:1707.07436 [cs.DM]
2. Brandstädt, A., Hundt, C.: Ptolemaic graphs and interval graphs are leaf powers. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) LATIN 2008. LNCS, vol. 4957, pp. 479–491. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78773-0_42
3. Calamoneri, T., Frangioni, A., Sinaimeri, B.: Pairwise compatibility graphs of caterpillars. Comput. J. **57**(11), 1616–1623 (2014)

4. Calamoneri, T., Frascaria, D., Sinaimeri, B.: All graphs with at most seven vertices are pairwise compatibility graphs. Comput. J. **56**(7), 882–886 (2013)
5. Calamoneri, T., Sinaimeri, B.: On pairwise compatibility graphs: a survey. SIAM Rev. **58**(3), 445–460 (2016)
6. Calamoneri, T., Petreschi, R.: On pairwise compatibility graphs having Dilworth number two. Theoret. Comput. Sci. **524**, 34–40 (2014)
7. Durocher, S., Mondal, D., Rahman, Md.S.: On graphs that are not PCGs. Theoret. Comput. Sci. **571**, 78–87 (2015)
8. Felsenstein, J.: Cases in which parsimony or compatibility methods will be positively misleading. Syst. Zool. **27**, 401–410 (1978)
9. Kearney, P., Munro, J.I., Phillips, D.: Efficient generation of uniform samples from phylogenetic trees. In: Benson, G., Page, R.D.M. (eds.) WABI 2003. LNCS, vol. 2812, pp. 177–189. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39763-2_14
10. Mehnaz, S., Rahman, M.S.: Pairwise compatibility graphs revisited. In: Proceedings of the International Conference on Informatics, Electronics Vision (ICIEV) (2013)
11. Salma, S.A., Rahman, Md.S.: Triangle-free outerplanar 3-graphs are pairwise compatibility graphs. J. Graph Algorithms Appl. **17**(2), 81–102 (2013)
12. Yanhaona, M.N., Bayzid, Md.S., Rahman, Md.S.: Discovering pairwise compatibility graphs. Discrete Math. Algorithms Appl. **2**(4), 607–623 (2010)
13. Yanhaona, M.N., Hossain, K.S.M.T., Rahman, Md.S.: Pairwise compatibility graphs. J. Appl. Math. Comput. **30**, 479–503 (2009)
14. Hossain, Md.I., Salma, S.A., Rahman, Md.S., Mondal, D.: A necessary condition and a sufficient condition for pairwise compatibility graphs. J. Graph Algorithms Appl. **21**(3), 341–352 (2017)

Efficient Unbounded Fault-Tolerant Aggregate Signatures Using Nested Cover-Free Families

Thais Bardini Idalino(✉) and Lucia Moura

University of Ottawa, Ottawa, Canada
{tbardini,lmoura}@uottawa.ca

Abstract. Aggregate signatures are used to create one short proof of authenticity and integrity from a set of digital signatures. However, one invalid signature in the set invalidates the entire aggregate, giving no information on which signatures are valid. Hartung et al. (PKC 2016) proposed a fault-tolerant aggregate signature scheme based on combinatorial group testing. Given a bound d on the number of invalid signatures, the scheme can determine which signatures are invalid, and guarantees a moderate increase on the size of the aggregate signature when there is an upper bound on the number n of signatures to be aggregated. However, for the case of unbounded n the constructions provided had constant compression ratio, i.e. the signature size grew linearly with n. In this paper we propose a solution to the unbounded scheme with increasing compression ratio for every d. In particular, for $d = 1$ the compression ratio is the best possible and meets the information theoretical bound.

Keywords: Aggregate signature · Fault-tolerance · Cover-free family
Digital signature · Combinatorial group testing

1 Introduction

Aggregate signature schemes allow us to combine a set of signatures into a single signature, which can be used as proof of integrity and authenticity of a possibly large set of data. This solution is useful specially for applications that manage a large quantity of data and digital signatures, since it can save on communication and storage, as well as improve the signature verification process. A few examples of such applications are outsourced databases [9], sensor networks [6], secure logging [7], certificate chains [1], vehicular communication [12], among others.

The verification of an aggregate signature outputs a positive result only if the entire set of signatures is valid. If we have at least one faulty signature in the set, the proof of integrity and authenticity of all the data involved is invalidated. This happens because when a set of signatures is aggregated into one, this operation does not preserve enough information in order to identify the exact set of invalid signatures. In order to solve this problem, Hartung et al. [2] propose a fault-tolerant scheme using d-cover-free families (d-CFFs). This scheme generates a

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 52–64, 2018.
https://doi.org/10.1007/978-3-319-94667-2_5

more expressive aggregate signature, that can tolerate up to d invalid signatures and identify all the valid ones.

Although the use of cover-free families is a practical solution for fault-tolerance in aggregated signatures, it sets a bound on the number of signatures n that can be aggregated. This number is not always known in advance; for example, secure logging applications can not always predict how many log entries will be saved, and outsourced databases not always have control on the amount of data that will be inserted. While a traditional aggregate scheme does not set such a bound, it also does not allow the identification of invalid signatures. This raises the need of an unbounded fault-tolerant aggregate scheme. In order to solve this problem, the authors in [2] present an unbounded scheme based on a *monotone* sequence of d-cover-free families. In an unbounded scheme, the concept of compression ratio is important. Consider the total number of signatures n and the size of the aggregate signature $s(n)$, the compression ratio is equals to $\rho(n)$ iff $\frac{n}{s(n)}$ is $\Theta(\rho(n))$. The explicit constructions of monotone families given in [2] yield aggregate signatures with length linear in n and thus *constant* compression ratio. This makes the scheme impractical for the unbounded case, even though the constant compression ratio can be chosen arbitrarily small. Intuitively, an increasing ratio is preferable since it implies a small aggregated signature. The authors pose as an open problem to find a better monotone family in order to achieve a more efficient unbounded scheme.

Contributions: In this work, we approach unbounded aggregate signatures by defining a more flexible sequence of $d-$CFFs, which we call a *nested* family, instead of monotone family defined in [2]. We show how to achieve unbounded aggregation in a general way by using nested families and give the corresponding aggregation algorithm. We provide explicit constructions of such families that yield unbounded aggregate schemes with better compression ratio than the constant ones previously known [2]. We give such constructions for general d (Theorems 9 and 11) and specific constructions for $d = 1$ (Theorem 7) and $d = 2$ (Theorem 12), which here can have a better compression ratio.

2 Background on Fault-Tolerant Schemes

In this section, we present the fault-tolerant aggregate signature scheme by Hartung et al. [2]. We summarize the concepts introduced by the authors [2, Sects. 1, 3 and 4], including the necessary formalization to contextualize our construction. The reader should refer to the original work [2] for a more detailed description. We also present background and recursive constructions of cover-free families.

2.1 Fault-Tolerant Aggregate Signature

Let $\{\sigma_1, \ldots, \sigma_n\}$ be a set of n signatures and let $C = \{(pk_1, m_1), \ldots, (pk_n, m_n)\}$ be their corresponding pairs of public key and message. A traditional aggregate

signature scheme consists of combining all n signatures together in one aggregate signature σ, which can be as small as a single digital signature [1]. By verifying only σ we can ensure the integrity and authenticity of the entire set C. If all signatures $\{\sigma_1, \ldots, \sigma_n\}$ are correctly formed from C, the signature verification outputs 1, but if at least one σ_i does not match its corresponding pair (pk_i, m_i), the verification outputs 0. In more details, Boneh et al. [1] define an aggregate signature scheme with four algorithms [1,2]:

1. **KeyGen**(1^κ) takes security parameter κ and creates a key pair (pk, sk).
2. **Sign**(sk, m) creates a signature for message m using secret key sk.
3. **Agg**$(C_1, C_2, \sigma_1, \sigma_2)$ takes two multisets of public key and message pairs C_1 and C_2 and their corresponding signatures σ_1 and σ_2, and outputs an aggregate signature σ that certifies the integrity and authenticity of $C = C_1 \cup C_2$.
4. **Verify**(C, σ) takes a multiset of public key and message pairs and its aggregate signature σ. Outputs 1 if the signature is valid and 0 otherwise.

The security of the scheme is based on the difficulty of an adversary to forge a signature of a chosen message after performing q signature queries to an oracle. More specifically, the aggregate scheme is (t, q, ϵ)-secure if there is no adversary $\mathcal{A}$ capable of winning the game with probability at least ϵ, performing at most q queries to the oracle, and running in time at most t [1,2].

In order to provide fault-tolerance, the signature verification needs to output a list of valid signatures instead of just 0 or 1, so the scheme should provide *list verification* instead of boolean verification. To describe this scheme, Hartung et al. [2] use the concepts of "claim" $c = (pk, m)$ as a tuple of message m and public key pk, and "claim sequence" C as a sequence of claims. A claim sequence requires an order among the claims, so each of its position i may contain one claim or a placeholder $\perp$. Two claim sequences C_1, C_2 are defined as exclusively mergeable if for all i, $C_1[i] = \perp$ or $C_2[i] = \perp$, and for a C_1 and C_2 of length k and l, with $k \geq l$, $C_1 \sqcup C_2 = (c_1, \ldots, c_k)$ is defined by

$$c_i = \begin{cases} C_1[i], \text{ if } C_2[i] = \perp, C_2[i] = C_1[i] \text{ or } i > l \\ C_2[i], \text{ otherwise.} \end{cases}$$

Let $C = (c_1, \ldots, c_n)$ be a claim sequence and $b \in \{0,1\}^n$ be a bit sequence that specifies a selection of indexes, then $C[b]$ denotes a subsequence of C with claims c_j for all j s.t. $b[j] = 1$. For more details, see Hartung et al. [2, Sect. 3]. The definition of list verification is given below.

Definition 1. *[Hartung et al. [2]] An aggregate signature scheme with list verification consists of four algorithms Σ:*

1. ***KeyGen**(1^κ) takes security parameter κ and creates a key pair (pk, sk).*
2. ***Sign**(sk, m) creates a signature for message m using secret key sk.*
3. ***Agg**$(C_1, C_2, \tau_1, \tau_2)$ takes two exclusively mergeable claim sequences C_1 and C_2 and their corresponding signatures τ_1 and τ_2, and outputs an aggregate signature τ that certifies the integrity and authenticity of the sequence $C = C_1 \sqcup C_2$.*

4. ***Verify***(C, τ) *takes a claim sequence* C *and its aggregate signature* τ *as input. Outputs the set of valid claims, which can be all the elements in* C *or even the empty set.*

Consider a claim sequence C with n claims, their corresponding signatures $\sigma_1, \ldots, \sigma_n$ with at most d invalid ones, and the aggregate signature τ. The aggregate signature scheme Σ is tolerant against d errors if $\Sigma.\mathbf{Verify}(C, \tau)$ outputs the set of claims that have valid signatures. Therefore, a *d-fault-tolerant* aggregate signature scheme is an aggregate signature scheme with list verification with a tolerance against d errors. From now on we will use σ to represent a standard aggregate signature, and τ for signatures of a fault-tolerant scheme.

Hartung et al. [2] uses d-CFFs to instantiate a generic fault-tolerant aggregate signature scheme. A d-CFF is a set system and can be represented by a $t \times n$ incidence binary matrix $\mathcal{M}$ (see Sect. 2.2 for more details). Given $\mathcal{M}$ and a set $\{\sigma_1, \ldots, \sigma_n\}$ of signatures to be aggregated, each column j represents a signature σ_j, and the rows of $\mathcal{M}$ indicate which signatures will be aggregated together. We are able to identify all valid signatures as long as the amount of invalid ones does not exceed a bound d.

Hartung et al. [2, Sect. 4] define a fault-tolerant aggregate signature scheme based on an ordinary aggregate signature scheme Σ that supports claims, claim sequences, and the empty signature λ as input. We denote $\mathcal{M}_i$ as row i of matrix $\mathcal{M}$, so $C[\mathcal{M}_i]$ represents the corresponding subsequence of a claim sequence C. This scheme inherits the security of Σ, with algorithms presented below:

1. $\mathbf{KeyGen}(1^\kappa)$ creates a key pair (pk, sk) using **KeyGen** from Σ and security parameter κ.
2. $\mathbf{Sign}(sk, m)$ receives a secret key and message, and outputs the signature given by $\Sigma.\mathbf{Sign}(sk, m)$.
3. $\mathbf{Agg}(C_1, C_2, \tau_1, \tau_2)$ takes two exclusive mergeable claim sequences C_1 and C_2 and corresponding signatures τ_1 and τ_2, and proceeds as follows:
   (a) If one or both of the claim sequences C_k ($k \in \{1, 2\}$) contain only one claim c, τ_k is an individual signature σ_k. We initialize σ_k as τ_k and expand it to a vector as follows, with j equals to the index of c in C_k:

$$\tau_k[i] = \begin{cases} \sigma_k, & \text{if } \mathcal{M}[i, j] = 1, \\ \lambda, & \text{otherwise} \end{cases} \quad \text{for } i = 1, \ldots t.$$

   (b) Once τ_1 and τ_2 are both vectors, we aggregate them, position by position, according to the incidence matrix $\mathcal{M}$:

$$\tau[i] = \Sigma.\mathbf{Agg}(C_1[\mathcal{M}_i], C_2[\mathcal{M}_i], \tau_1[i], \tau_2[i]).$$

   (c) Output τ, which certifies the integrity and authenticity of the sequence $C = C_1 \sqcup C_2$.
4. $\mathbf{Verify}(C, \tau)$ takes a set of public key and message pairs and the aggregate signature τ, and outputs the set of valid claims. Computes $b_i = \Sigma.\mathbf{Verify}(C[\mathcal{M}_i], \tau[i])$ for each $1 \leq i \leq t$ and outputs the set of valid claims consisting of the union of each $C[\mathcal{M}_i]$ such that $b_i = 1$.

The following theorems are from Hartung et al. [2, Sect. 4] and address the security and correctness of the scheme. For details regarding their proofs, see [2].

Theorem 1 (Hartung et al. [2]). *Let Σ be the aggregate signature scheme with list verification presented above. If Σ is based on a d-CFF, then it is correct and tolerant against up to d errors.*

Theorem 2 (Hartung et al. [2]). *If Σ is a (t, q, ϵ)-secure aggregate signature scheme, then the aggregate signature scheme with list verification above is (t', q, ϵ)-secure, with t' approximately the same as t.*

The following example consists of $n = 10$ signatures aggregated according to the 1-CFF$(5, 10)$ matrix $\mathcal{M}$, which allows us to identify all valid signatures as long as we have at most one invalid. For instance, if σ_1 is invalid, $\tau[1]$ and $\tau[2]$ will fail, but $\tau[3], \tau[4], \tau[5]$ prove the validity of $\sigma_i, 2 \leq i \leq 10$.

$$\mathcal{M} = \begin{pmatrix} 1\,1\,1\,1\,0\,0\,0\,0\,0\,0 \\ 1\,0\,0\,0\,1\,1\,1\,0\,0\,0 \\ 0\,1\,0\,0\,1\,0\,0\,1\,1\,0 \\ 0\,0\,1\,0\,0\,1\,0\,1\,0\,1 \\ 0\,0\,0\,1\,0\,0\,1\,0\,1\,1 \end{pmatrix} \rightarrow \begin{array}{r} \tau[1] = Agg(\sigma_1, \sigma_2, \sigma_3, \sigma_4) \\ \tau[2] = Agg(\sigma_1, \sigma_5, \sigma_6, \sigma_7) \\ \tau[3] = Agg(\sigma_2, \sigma_5, \sigma_8, \sigma_9) \\ \tau[4] = Agg(\sigma_3, \sigma_6, \sigma_8, \sigma_{10}) \\ \tau[5] = Agg(\sigma_4, \sigma_7, \sigma_9, \sigma_{10}) \end{array}$$

The idea of fault tolerance in signature aggregation using CFFs appeared independently in the master's thesis of the first author as *level-d signature aggregation* [3, Chap. 5]. A related application of CFFs to modification tolerant digital signatures can be found in [4].

2.2 Cover-Free Family Constructions

Cover-free families (CFFs) are combinatorial structures studied in the context of combinatorial group testing, and frequently used in scenarios where we need to test a set of n elements to identify up to d invalid ones. We use them to combine these elements into a few groups, and test the groups instead of each element.

Definition 2. *A set system $\mathcal{F} = (X, \mathcal{B})$ consists of a set $X = \{x_1, \ldots, x_t\}$ with $|X| = t$, and a collection $\mathcal{B} = \{B_1, \ldots, B_n\}$ with $B_i \subseteq X, 1 \leq i \leq n$, and $|\mathcal{B}| = n$. A d-cover-free family, denoted $d-CFF(t, n)$, consists of a set system such that for any subset $B_{i_0} \in \mathcal{B}$ and any other d subsets $B_{i_1}, \ldots, B_{i_d} \in \mathcal{B}$, we have*

$$B_{i_0} \nsubseteq \bigcup_{j=1}^{d} B_{i_j} \tag{1}$$

We can represent $\mathcal{F}$ as a $t \times n$ binary incidence matrix $\mathcal{M}$ by considering the characteristic vectors of subsets in $\mathcal{B}$ as columns of $\mathcal{M}$. More precisely, $\mathcal{M}_{i,j} = 1$ if $x_i \in B_j$, and $\mathcal{M}_{i,j} = 0$ otherwise. We will interchangeably say that $\mathcal{M}$ is d-CFF when its corresponding set system is d-CFF. Note that if $\mathcal{M}$ is d-CFF, then a matrix obtained by row and column permutations is also d-CFF.

An equivalent definition of d-CFF is based on the existence of permutation submatrices of dimension $d + 1$ [8]. A permutation matrix is an $n \times n$ binary matrix with exactly one "1" per row and per column, in other words, it is obtained by permuting the rows of the identity matrix.

Proposition 1. *A matrix $\mathcal{M}$ is d-CFF if and only if any set of $d+1$ columns contains a permutation sub-matrix of dimension $d+1$.*

The next propositions state relationships between sub-matrices with respect to d-CFF properties. Their proofs follow directly from Definition 2.

Proposition 2. *Let $\mathcal{M}$ be a matrix and let $\mathcal{M}'$ be a sub-matrix of $\mathcal{M}$ formed by some of its columns. If $\mathcal{M}$ is d-CFF, then $\mathcal{M}'$ is also d-CFF.*

Proposition 3. *Let $\mathcal{M}$ be a matrix and let $\mathcal{M}'$ be a sub-matrix of $\mathcal{M}$ formed by some of its rows. If $\mathcal{M}'$ is d-CFF, then $\mathcal{M}$ is d-CFF.*

When considering $d = 1$, a $1-$CFF allows us to identify up to one invalid element. For this particular case, there is a construction using Sperner's theorem that yields minimum possible number of rows.

Theorem 3 *(Sperner [11]).* *Let $\mathcal{B}$ be a collection of subsets of $\{1, \ldots, t\}$ such that $B_1 \not\subseteq B_2$ for all $B_1, B_2 \in \mathcal{B}$. Then $|\mathcal{B}| \leq \binom{t}{\lfloor t/2 \rfloor}$. Moreover, equality holds when $\mathcal{B}$ is the collection of all the $\lfloor t/2 \rfloor$-subsets of $\{1, \ldots, t\}$.*

Corollary 1. *Given n and $d = 1$, there exists a 1-CFF(t,n) matrix with $t = \min\{s : \binom{s}{\lfloor s/2 \rfloor} \geq n\}$.*

Proof. Build each column of the matrix from the characteristic vector of a distinct $\lfloor t/2 \rfloor$-subset of a t-set. Since this forms a Sperner family of subsets, Property (1) for $d = 1$ holds. □

The value t grows as $\log_2 n$ as $n \to \infty$, which meets the information theoretical lower bound, yielding an optimal construction.

Other constructions of CFF exist for larger d; see Zaverucha and Stinson [13, Sect. 3.2] for a discussion on how other combinatorial objects yield good CFF methods depending on the relation of d and n. In particular, Porat and Rothschild [10] give a construction that yields $t = c(d+1)^2 \log n$ for a constant c, which for fixed d is optimal in terms of meeting a lower bound $\Theta(\log n)$ (see [13, Theorem 3]). Next we give generalizations of two constructions by Li et al. [5, Theorems 3.4 and 3.5] that allows us to build larger $d-$CFFs from smaller ones.

Definition 3 (Kronecker product). *Let A_k be an $m_k \times n_k$ binary matrix, for $k = 1, 2$. The product $P = A_1 \otimes A_2$ is a binary matrix such that*

$$P = \begin{pmatrix} P_{1,1} & \ldots P_{1,n_1} \\ \vdots & \vdots \\ P_{m_1,1} & \ldots P_{m_1,n_1} \end{pmatrix} \quad \textit{where} \quad P_{i,j} = \begin{cases} A_2, & \textit{if } A_{1_{i,j}} = 1 \\ \mathbf{0}, & \textit{otherwise} \end{cases}$$

Where **0** is the matrix of all zeroes with same dimension as A_2. The following theorem generalizes a construction by Li et al. [5] given for $d = 2$.

Theorem 4. *Let A_1 be a $d-CFF(t_1, n_1)$ and A_2 be a $d-CFF(t_2, n_2)$, then $C = A_1 \otimes A_2$ is a $d-CFF(t_1 t_2, n_1 n_2)$.*

Next we present a construction of d-CFFs based on results by Li et al. [5] for $d = 2$. This construction gives a better result than the one from Theorem 4 for the cases where $s < \frac{t_1 t_2 - t_2}{t_1}$.

Construction 1. *Let $d \geq 2$, A_1 be a $d-CFF(t_1, n_1)$, A_2 be a $d-CFF(t_2, n_2)$, and B be a $(d-1)-CFF(s, n_2)$. Create a matrix $P = B \otimes A_1$ as in Definition 3. This results in P with n_2 "blocks" of n_1 columns each. For each column in block i, append the ith column of A_2, for $1 \leq i \leq n_2$. Call* Const1(A_1, A_2, B) *the matrix obtained.*

Theorem 5. *Let A_1 be a $d-CFF(t_1, n_1)$, A_2 be a $d-CFF(t_2, n_2)$, and B be a $(d-1)-CFF(s, n_2)$. Then* C := ***Const1****(A_1, A_2, B) is $d-CFF(st_1 + t_2, n_1 n_2)$.*

As a corollary, it is possible to obtain a previous result by Li et al. [5] for the specific case of $d = 2$.

Corollary 2. *[5, Theorem 3.5] Suppose there exists a $2-CFF(t_1, n_1)$ and a $2-CFF(t_2, n_2)$, then there exists a $2-CFF(st_1 + t_2, n_1 n_2)$ for any s satisfying $\binom{s}{\lfloor \frac{s}{2} \rfloor} \geq n_2$.*

3 Our General Unbounded Scheme

By using $d-$CFFs for fault-tolerant aggregate signatures, we set a bound on the number of signatures n that can be aggregated, which may not be known in advance. Applications such as secure logging usually cannot predict the amount of signed logs that will be generated, and dynamic databases may not be able to predict the amount of data that will be inserted. These unbounded applications require a sequence of $d-$CFFs that allows the increase of n as necessary. Several of these applications also deal with a large amount of signatures, and it may not be possible to save each one of them individually. So besides requiring increasing size, the $d-$CFF should also take into consideration that once aggregated, the individual signatures may not be available anymore. Moreover, we want to use the best constructions for a specific number of signatures n, corresponding to an increasing compression ratio as n grows. This raises the need of a sequence of $d-$CFFs to support all these requirements.

In order to address this problem, Hartung et al. [2] propose the notion of a fault-tolerant unbounded scheme based on what they call a *monotone* family of d-CFFs. It consists of using a CFF incidence matrix $\mathcal{M}^{(1)}$ until its maximum n is achieved, and then jump to the next matrix. Each new matrix $\mathcal{M}^{(l+1)}$ contains the previous ones $\mathcal{M}^{(1)}, \ldots, \mathcal{M}^{(l)}$ in a sequence that presents a monotonicity property. However, the constructions they provide for monotone families yield

aggregate signature with length linear in n and constant compression ratio, which makes the scheme impractical for the unbounded case. Intuitively, an unbounded fault-tolerant aggregate scheme should provide increasing compression ratio as we increase n, since it reflects a smaller increase on the aggregate signature size.

In this section, we extend the notion of unbounded aggregation and suggest a more flexible sequence of $d-$CFFs, called *nested* family. We claim that even without the monotone property we can keep the characteristic of being able to discard individual signatures after they are aggregated. In fact, by using a nested family we can offer an infinite sequence of d-cover-free families with increasing compression ratio. The new definition of nested families is presented below.

In the remaining of the paper, an infinite sequence $a_1, a_2, \ldots$ is compactly denoted as $(a_l)_l$ for sets and $(a^{(l)})_l$ for matrices.

Definition 4. *Let $(\mathcal{M}^{(l)})_l$ be a sequence of incidence matrices of d-cover-free families $(\mathcal{F}_l)_l = (X_l, \mathcal{B}_l)_l$, where the number of rows and columns of $\mathcal{M}^{(l)}$ are denoted by rows(l) and cols(l), respectively. $(\mathcal{M}^{(l)})_l$ is a* nested family *of incidence matrices of d-CFFs, if $X_l \subseteq X_{l+1}$, rows(l) $\leq$ rows(l+1), and cols(l) $\leq$ cols(l+1), and*

$$\mathcal{M}^{(l+1)} = \begin{pmatrix} \mathcal{M}^{(l)} & Y \\ Z & W \end{pmatrix}$$

where each row of Z is one of the rows of $\mathcal{M}^{(l)}$, or a row of all zeros, or a row of all ones.

Note that the definition of monotone family by Hartung et al. [2] is a special case of nested family, where $Z = 0$. The authors use monotone families to achieve unbounded aggregation in the following way. For each $1 \leq i \leq cols(l)$, if $B_i \in \mathcal{B}_l$ and $D_i \in \mathcal{B}_{l+1}$, then $B_i = D_i$. In the case of nested families, instead of $B_i = D_i$ we get $B_i \subseteq D_i$ for all $1 \leq i \leq cols(l)$. The additional property requiring that the rows of Z must repeat rows of $\mathcal{M}^{(l)}$, or be trivial, is what allows us to be able to only need previous aggregations and not original signatures. We observe that subsequences of nested families are also nested families.

Our unbounded fault-tolerant aggregate signature scheme with nested families is defined by the following algorithms. Note that **KeyGen** and **Sign** are equal to the algorithms given in page 4. We also create a new position $\tau[0]$ in the aggregate signature τ, which holds a full aggregation of all signatures considered up to that point. Let $(\mathcal{M}^{(l)})_l$ be a nested family of incidence matrices of d-CFFs and let Σ be a simple aggregate signature scheme that supports claim sequences, claim placeholders, and the empty signature λ.

1. **KeyGen**(1^κ) creates a key pair (pk, sk) using **KeyGen** from Σ and security parameter κ.
2. **Sign**(sk, m) receives a secret key and message, and outputs the signature given by Σ.**Sign**(sk, m).
3. **Agg**$(C_1, C_2, \tau_1, \tau_2)$ takes two exclusive mergeable claim sequences C_1 and C_2 and corresponding signatures τ_1 and τ_2, and outputs the aggregate signature τ, where $|\tau| = \max\{|\tau_1|, |\tau_2|\}$.

(a) Let n_k be the dimension of C_k for $k = 1, 2$, and assume w.l.o.g. that $n_1 \leq n_2$. Determine l_k such that $\text{cols}(\mathcal{M}^{(l_k-1)}) < n_k \leq \text{cols}(\mathcal{M}^{(l_k)})$, and denote by $t_k = \text{rows}(\mathcal{M}^{(l_k)})$, $k = 1, 2$. Note that $l_1 \leq l_2$ and take $\mathcal{M}$ as the submatrix of $\mathcal{M}^{(l_2)}$ consisting of the first n_2 columns. Note that $\mathcal{M} = \left(\begin{smallmatrix} \mathcal{M}^{(l_1)} & Y \\ Z & W \end{smallmatrix}\right)$, for some matrices Z, Y, W satisfying the "nesting" properties of Definition 4. For this aggregation, $\mathcal{M}$ will be the d-CFF matrix that plays the same role as the fixed matrix used in the bounded scheme in page 4.

(b) If one or both of the claim sequences C_k ($k \in \{1, 2\}$) contain only one claim c, τ_k is an individual signature σ_k. We expand τ_k to a vector as follows, with j equals to the index of c in C_k:

$$\tau_k[i] = \begin{cases} \sigma_k, & \text{if } i = 0 \text{ or } (\mathcal{M}[i, j] = 1 \text{ and } 1 \leq i \leq t_k), \\ \lambda, & \text{otherwise.} \end{cases}$$

(c) Once τ_1 and τ_2 are both vectors, we aggregate them position by position according to $\mathcal{M}$. Note that by the nested family definition we have three types of row index i depending on the row type of Z: a row of zeros, where $\mathcal{M}[i, 1] = \ldots = \mathcal{M}[i, n_1] = 0$ (Type 0); a row of ones, where $\mathcal{M}[i, 1] = \ldots = \mathcal{M}[i, n_1] = 1$ (Type 1); and a repeated row r of $\mathcal{M}^{(l_1)}$, where $\mathcal{M}[i, 1] = \mathcal{M}^{(l_1)}[r, 1] = \ldots = \mathcal{M}[i, n_1] = \mathcal{M}^{(l_1)}[r, n_1]$ (Type 2 (r)). First we expand C_1 to $\overline{C}_1$ having the same dimension as C_2, i.e. $\overline{C}_1[i] = C_1[i]$ for $1 \leq i \leq n_1$, and $\overline{C}_1[i] = \perp$ for $n_1 + 1 \leq i \leq n_2$, then we proceed as follows.

$$\tau[0] = \Sigma.\mathbf{Agg}(\overline{C}_1, C_2, \tau_1[0], \tau_2[0])$$

For $i = 1, \ldots t_1$:

$$\tau[i] = \Sigma.\mathbf{Agg}(\overline{C}_1[\mathcal{M}_i], C_2[\mathcal{M}_i], \tau_1[i], \tau_2[i])$$

For $i = t_1 + 1, \ldots, t_2$:

$$\tau[i] = \begin{cases} \tau_2[i], & \text{if } i \text{ is Type 0,} \\ \Sigma.\mathbf{Agg}(\overline{C}_1[\mathcal{M}_i], C_2[\mathcal{M}_i], \tau_1[0], \tau_2[i]), & \text{if } i \text{ is Type 1,} \\ \Sigma.\mathbf{Agg}(\overline{C}_1[\mathcal{M}_i], C_2[\mathcal{M}_i], \tau_1[r], \tau_2[i]), & \text{if } i \text{ is Type 2 (r).} \end{cases}$$

Output τ.

4. **Verify**(C, τ) takes a set of public key and message pairs and the aggregate signature τ and outputs the valid claims. If $\Sigma.\mathbf{Verify}(C, \tau[0]) = 1$, output all claims, otherwise compute $b_i = \Sigma.\mathbf{Verify}(C[\mathcal{M}_i], \tau[i])$ for each $1 \leq i \leq t_2$ and output the set of valid claims consisting of the union of each $C[\mathcal{M}_i]$ such that $b_i = 1$.

The correctness of the aggregation and verification algorithms comes from the fact that the matrices used are d-CFF. For the aggregation algorithm we just need to verify that the aggregated signature computed in step (c) yields

the same results as if $\mathcal{M}^{(l_2)}$ was used directly on the original signatures. The security of the scheme comes from Theorem 2, which relies on the security of the underlying aggregate scheme Σ.

In the next section, we give three explicit constructions of nested families that allow us to achieve unbounded aggregation with increasing compression ratio for $d = 1$, $d = 2$, and general values of d.

4 Construction of Unbounded Schemes with Non Constant Compression Ratio

Now we aim to construct a nested family of incidence matrices as shown in Definition 4, where we can increase n as necessary while avoiding to save every individual signature for further use. In this section, we propose explicit constructions of nested families for the cases where $d = 1$, $d = 2$, and for general values of d. We note that all sequences of CFF given are constructive, as they rely on ingredients that can be constructed explicitly by known methods. Proofs are omitted here, and will be included in an extended version of this paper.

4.1 Nested Family for d = 1

As specified in Corollary 1, we can build a $1-$CFF $\mathcal{F} = (X, \mathcal{B})$ from a set of elements $X = \{x_1, \ldots, x_t\}$ and a collection $\mathcal{B}$ of $\lfloor t/2 \rfloor-$subsets of X, with $|\mathcal{B}| = n$. Given n, we need to choose a minimum t such that $n \leq \binom{t}{\lfloor t/2 \rfloor}$. From the binomial properties we can see that one specific t may be suitable for a few consecutive n values, as an example for $n = 7, 8, 9, 10$ we can use $t = 5$. We want to achieve unbounded aggregation by constructing a nested family with increasing n, so larger n values can be achieved by increasing t.

Now we focus on constructing a new $1-$CFF $\mathcal{F}_t = (X_t, \mathcal{B}_t)$ from a smaller $\mathcal{F}_{t-1} = (X_{t-1}, \mathcal{B}_{t-1})$, in order to obtain a sequence of nested families. Let $X_t = \{1, \ldots, t\}$ and let $\mathcal{B}_t$ be all distinct $\lfloor \frac{t}{2} \rfloor-$subsets of elements from X_t. We consider small increments of n that increase t by one, i.e. $X_t = X_{t-1} \cup \{t\}$, and $\mathcal{B}_t$ is generated by considering two cases according to the parity of t, as shown below.

Since the order of the elements is important for nested families, we represent the family as a tuple of sets. Let $n_l = \binom{l}{\lfloor \frac{l}{2} \rfloor}$, and $\mathcal{C}_k^n$ be the list of all k-subsets from $\{1, \ldots, n\}$ in lexicographical order. The recursive definition of $\mathcal{B}_t$ is:

$$\mathcal{B}_2 = [\{1\}, \{2\}],$$

$$\mathcal{B}_t = \begin{cases} [\mathcal{B}_{t-1}[1], \ldots, \mathcal{B}_{t-1}[n_{t-1}], \mathcal{C}^{t-1}_{\lfloor t/2 \rfloor - 1}[1] \cup \{t\}, \ldots, \mathcal{C}^{t-1}_{\lfloor t/2 \rfloor - 1}[n_t] \cup \{t\}], & t \text{ odd}, \\ [\mathcal{B}_{t-1}[1] \cup \{t\}, \ldots, \mathcal{B}_{t-1}[n_{t-1}] \cup \{t\}, \mathcal{C}^{t-1}_{t/2}[1], \ldots, \mathcal{C}^{t-1}_{t/2}[n_t]], & t \text{ even}, \end{cases}$$

for $t > 2$.

Theorem 6. *The sequence $(X_t, \mathcal{B}_t)_t$ is a nested family, and each family in the sequence is $1-$CFF.*

It is easy to see that the property holds for larger increases of t as well, where each row of Z in this construction consists of all zeros or all ones. In other words, taking sub-sequences of $(X_t, \mathcal{B}_t)_t$ also gives a nested family.

Theorem 7. *Let $(\mathcal{M}^{(l)})_l$ be the nested family defined in Theorem 6. This sequence has a compression ratio $\rho(n) = \frac{n}{\log_2 n}$.*

Note that this compression ratio meets the information theoretical bound.

4.2 Nested Family for $d \geq 2$

In this section, we give three classes of constructions of nested families which for fixed d gives $\rho(n) \to \infty$ as $n \to \infty$. Theorems 9 and 11 give specific compression ratios obtained for the cases of $d \geq 2$. The one from Theorem 11 is asymptotically better, but for different constants and ranges of n, either one may be more suitable. Theorem 12 gives yet a better asymptotic ratio for the case $d = 2$.

Theorem 8 gives nested families via Theorem 4, and Theorem 9 provides its corresponding ratio.

Theorem 8. *Let $\mathcal{M}$ be the incidence matrix of a $d-CFF(t, n)$ (wlog we require $\mathcal{M}_{1,1} = 1$), and set $\mathcal{M}^{(1)} = \mathcal{M}$. We define $\mathcal{M}^{(l)} = \mathcal{M} \otimes \mathcal{M}^{(l-1)}$ for $l \geq 2$. Then $(\mathcal{M}^{(l)})_l$ is a nested family of incidence matrices of d-CFFs.*

Theorem 9. *Let $d \geq 2$. Let $(\mathcal{M}^{(l)})_l$ be the nested family defined in Theorem 8 using a d-CFF$(\bar{t}, \bar{n})$ matrix $\mathcal{M}$ with $\bar{n} > \bar{t} > 1$. Then, the sequence has increasing compression ratio $\rho(n) = \frac{n}{n^{1/c}} = n^{1-1/c}$, for $c = \log_{\bar{t}} \bar{n} > 1$ (c depending on d).*

In Theorem 10, we show how to use the construction from Theorem 5 to build a sequence of nested families for general d. In order to achieve a nested family, we instantiate the construction with a single d-CFF(t_1, n_1) $\mathcal{M}^{(1)}$ used as both A_1 and A_2, and recursively apply the construction until we achieve the desired number of signatures n. Then, in Theorems 11 and 12 we give compression ratios by using specific d-CFF matrices.

Theorem 10. *Let $\mathcal{M}$ be a $d-CFF(t_1, n_1)$ matrix and set $\mathcal{M}^{(1)} = \mathcal{M}$. Let B_i be a $(d-1)$-CFF(s_i, n_i) matrix (wlog we require $B_{i_{1,1}} = 1$), for each $i \geq 1$. We recursively define $\mathcal{M}^{(l)} = \textbf{\textit{Const1}}(\mathcal{M}^{(l-1)}, \mathcal{M}^{(l-1)}, B_{l-1})$, for $l \geq 2$ and* Const1 *as defined in Construction 1. The sequence of matrices $(\mathcal{M}^{(l)})_l$ is a nested family of incidence matrices of d-CFFs.*

Theorem 11. *Let $d \geq 2$. Let $(\mathcal{M}^{(l)})_l$ be the nested family defined in Theorem 10 using a d-CFF$(\bar{t}, \bar{n})$ matrix $\mathcal{M}$ with $\bar{n} > \bar{t}$. Then, there exists a $(d-1)$-CFF B_l such that the sequence $(\mathcal{M}^{(l)})_l$ has increasing ratio $\rho(n) = \frac{n}{(b \log_2 n)^{\log_2 \log_2 n + D}}$ for constants $b = \frac{2d^2 \ln \bar{n}}{\log_2 \bar{n}}$, and $D = 1 - \log_2 \log_2 \bar{n}$.*

The next theorem improves the ratio from Theorem 11 for $d = 2$ by using optimal 1-CFF constructions given by Corollary 1.

Theorem 12. *Let $(\mathcal{M}^{(l)})_l$ be the nested family defined in Theorem 10 using a 2-CFF$(\bar{t}, \bar{n})$ matrix $\mathcal{M}$ with $\bar{n} > \bar{t}$, and each B_i is a 1-CFF given by Corollary 1 for $n = \bar{n}^{2^{i-1}}$, $i \geq 1$. Then, the sequence $(\mathcal{M}^{(l)})_l$ has increasing ratio $\rho(n) = \frac{n}{(2\log_2 n)^{\log_2 \log_2 n + D}}$, for constant $D = 1 - \log_2 \log_2 \bar{n}$.*

5 Final Remarks and Open Problems

In this work we define a sequence of cover-free families, called *nested* families, which generalizes monotone families introduced by Hartung et al. [2]. While it may be difficult to build a monotone family with increasing compression ratio, we show that this is possible by using nested families. We show how to achieve unbounded aggregation in general with nested families, and we also give explicit constructions for the cases of $d = 1$, $d = 2$, and general values of d. We were able to meet the information theoretical bound for $d = 1$, and good compression ratios for all values of d. We believe the concept of nested cover-free families can also be of important interest in other applications of cover-free families.

We observe that as n increases in our constructions, it would be desirable to increase the threshold value d as well. However, it is not hard to show that there can not be nested families with increasing d. We would like to investigate other sequences of CFFs with increasing d.

Acknowledgments. Thais Bardini Idalino acknowledges funding granted from CNPq-Brazil [233697/2014-4]. Lucia Moura was supported by an NSERC discovery grant.

References

1. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: EUROCRYPT 2003, pp. 416–432 (2003)
2. Hartung, G., Kaidel, B., Koch, A., Koch, J., Rupp, A.: Fault-tolerant aggregate signatures. In: Public-Key Cryptography - PKC 2016, pp. 331–356 (2016)
3. Idalino, T.B.: Using combinatorial group testing to solve integrity issues. Master's thesis, Universidade Federal de Santa Catarina, Brazil (2015)
4. Idalino, T.B., Moura, L., Custódio, R.F., Panario, D.: Locating modifications in signed data for partial data integrity. Inf. Process. Lett. **115**(10), 731–737 (2015)
5. Li, P.C., van Rees, G.H.J., Wei, R.: Constructions of 2-cover-free families and related separating hash families. J. Comb. Des. **14**(6), 423–440 (2006)
6. Li, Z., Gong, G.: Data aggregation integrity based on homomorphic primitives in sensor networks. In: Nikolaidis, I., Wu, K. (eds.) ADHOC-NOW 2010. LNCS, vol. 6288, pp. 149–162. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14785-2_12
7. Ma, D.: Practical forward secure sequential aggregate signatures. In: ASIACCS 2008, pp. 341–352. ACM (2008)
8. Macula, A.J.: A simple construction of d-disjunct matrices with certain constant weights. Discrete Math. **162**(1), 311–312 (1996)
9. Mykletun, E., Narasimha, M., Tsudik, G.: Authentication and integrity in outsourced databases. ACM Trans. Storage **2**, 107–138 (2006)

10. Porat, E., Rothschild, A.: Explicit nonadaptive combinatorial group testing schemes. IEEE Trans. Inf. Theory **57**, 7982–7989 (2011)
11. Sperner, E.: Ein Satz über Untermengen einer endlichen Menge. Mathematische Zeitschrift **27**, 544–548 (1928)
12. Wasef, A., Shen, X.: ASIC: aggregate signatures and certificates verification scheme for vehicular networks. In: GLOBECOM 2009, pp. 1–6 (2009)
13. Zaverucha, G.M., Stinson, D.R.: Group testing and batch verification. In: ICITS 2009, pp. 140–157 (2009)

Minimum Polygons for Fixed Visibility VC-Dimension

Moritz Beck(✉) and Sabine Storandt

Institut für Informatik, Julius-Maximilians-Universität Würzburg, Würzburg, Germany
{beck,storandt}@informatik.uni-wuerzburg.de

Abstract. Motivated by the art gallery problem, the visibility VC-dimension was investigated as a measure for the complexity of polygons in previous work. It was shown that simple polygons exhibit a visibility VC-dimension of at most 6. Hence there are 7 classes of simple polygons w.r.t. their visibility VC-dimension. The polygons in class 0 are exactly the convex polygons. In this paper, we strive for a more profound understanding of polygons in the other classes. First of all, we seek to find minimum polygons for each class, that is, polygons with a minimum number of vertices for each fixed visibility VC-dimension d. Furthermore, we show that for $d < 4$ the respective minimum polygons exhibit only few different visibility structures, which can be represented by so called visibility strings. On the practical side, we describe an algorithm that computes the visibility VC-dimension of a given polygon efficiently. We use this tool to analyze the distribution of the visibility VC-dimension in different kinds of randomly generated polygons.

1 Introduction

One of the most widely known and well studied visibility problems in algorithmic geometry is the *art gallery problem*. Given a polygon (the art gallery footprint) it asks for the minimum sized set of points (the guards) in the polygon that collectively see the entire polygon. There are many variations of the art gallery problem, for example depending on whether guards are only allowed on vertices of the polygon, only on the edges, or also inside the polygon; and whether the polygons are simple or contain holes. In [1,2] it was proven that all these variants are APX-hard, so an approximation ratio better than some fixed constant is unlikely to be obtainable.

In [3] a deterministic $\mathcal{O}(\log n)$-approximation algorithm was described for vertex and edge guards which transforms an art gallery problem to a set cover problem. Improvements over this approximation ratio were made by taking the geometric information into account: In [4], it was shown that the set systems of the respective set cover instances exhibit a bounded *VC-dimension*. For set systems with VC-dimension d, Brönnimann and Goodrich [5] introduced an algorithm based on ϵ-net theory which provides an $O(d \log(d \cdot \mathrm{OPT}))$-approximation

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 65–77, 2018.
https://doi.org/10.1007/978-3-319-94667-2_6

where OPT denotes the size of an optimal solution. Hence for constant d, a $\mathcal{O}(\log OPT)$-approximation is achieved.

So the smaller d, the better the approximation ratio. This motivates to study the visibility VC-dimension of polygons. Intuitively, the visibility VC-dimension measures the complexity of the visibility structure of a polygon.

Definition 1 (Visibility VC-Dimension). *Given a simple polygon with n vertices, a vertex* sees *(or* is visible from*) another vertex if the straight line segment between those vertices does not intersect the exterior of the polygon. A subset S of the polygon vertices is called* shattered *if for every subset S' of S there is a vertex v such that this vertex sees every vertex in S' but sees none of the vertices in $S \setminus S'$. The* visibility VC-dimension *d of the polygon is the size of its largest shattered subset.*

We call the vertices in S also the *VC-points* and v itself the *viewpoint* for the subset S'. Note that VC-points and viewpoints are not necessarily disjunct. Figure 1 illustrates all these concepts.

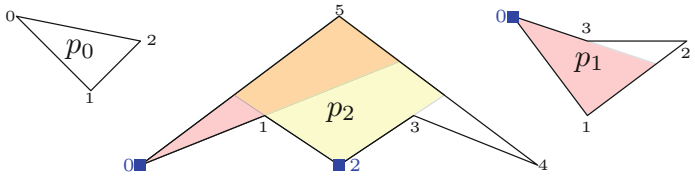


Fig. 1. Polygons p_d with VC-dimension $d \leq 2$. The blue vertices are the VC-points. In p_2 the vertex 0 sees itself but not 2, the vertex 3 is a viewpoint for the subset $S' = \{2\}$, the vertex 5 for $S' = \{0, 2\}$ and vertex 4 sees neither 0 nor 2. (Color figure online)

The visibility VC-dimension of simple polygons is defined as the maximum visibility VC-dimension over all simple polygons. For viewpoints being restricted to polygon vertices or edges, it was proven in [6] that the visibility VC-dimension of simple polygons is 6. For individual polygons the VC-dimension can assume any integer between 0 and 6. This induces seven classes of simple polygons. In this paper, we focus on studying minimum polygons within each class.

1.1 Related Work

In [4] it was proven that the visibility VC-dimension for simple polygons (allowing viewpoints to lie anywhere in the polygon) is at most 23 and an example polygon with VC-dimension 6 was provided. The upper bound was lowered to 14 by Gilbers and Klein [7]. For the case when the viewpoints are restricted to the boundary of the polygon the upper bound was consecutively lowered to 7 by Gilbers [8] and 6 by Gibson et al. [6]. A matching lower bound was presented in [9]. Hence the VC-dimension of simple polygons is known to be exactly 6.

Interestingly these bounds also hold for x-monotone polygons [9]. A polygon is x-monotone if the intersection between the polygon and any vertical line is connected. Somehow surprisingly, this structural limitation does not improve the visibility VC-dimension.

1.2 Contribution and Outline

In Sect. 2, we provide examples of minimum polygons for each visibility VC-dimension up to $d = 5$; along with the proofs that smaller polygons can not be members of the respective classes. Somehow surprisingly, for $d = 3, 4, 5$ the respective minimum polygons consist of exactly 2^d vertices which matches the trivial lower bound. For $d = 6$, we show that a minimum polygon consists of $64 \leq n \leq 78$ vertices and we conjecture that there also $n = 2^d = 64$ holds.

In Sect. 3, we further investigate the visibility structure of minimum polygons by abstracting from the geometry. We define so called visibility strings which capture the collocation of VC-points and the numbers of VC-points visible from the viewpoints. For $d < 4$, we prove that only few different visibility strings occur; for example, for $d = 3$ only two.

In Sect. 4, practical algorithms to compute the visibility VC-dimensions of a given polygon are discussed. This is complemented with experimental results in Sect. 5. There we compare the distribution of the visibility VC-dimension classes for different kinds of randomly generated polygons, including ones created by the 2-Opt method and Quick Star. We show that the distributions differ significantly in dependency of the creation method, and we furthermore devise a new polygon generator which is more likely to produce polygons with a high visibility VC-dimension. As such polygons are likely to be 'hard' instances for the art gallery problem, the generator might be useful to create sensible benchmark sets for guard placement algorithms.

2 Minimum Polygons

Given a VC-dimension d there is a trivial lower bound for the number of vertices a polygon of this VC-dimension must have: A shattered subset S of size d has 2^d subsets S'. For every of these subsets there has to be a vertex that sees S' but not $S \setminus S'$. So the number of viewpoints/vertices n_d is at least 2^d. The smallest simple polygons are triangles, so another trivial lower bound is 3 vertices. Concave spots are needed to get a VC-dimension higher than 0 as there has to be a viewpoint that does not see any VC-point. As all triangles are convex, a polygon with positive VC-dimension needs to have at least 4 vertices. We observe that indeed one concave spot is sufficient to get a VC-dimension of 1 (see Fig. 1, right), hence we deduce $n_1 = 4$. Table 1 summarizes all our derived bounds for the number of vertices of minimum polygons. The results for $d \geq 2$ are explained in more detail below.

Table 1. The minimum number of vertices required to get a certain VC-dimension.

VC-dimension d	0	1	2	3	4	5	6
Minimum vertex count n_d	3	4	6	8	16	32	$64 \leq n_6 \leq 78$

2.1 Minimum Polygons for VC-Dimension 2

For polygons with VC-dimension $d = 2$ actually more than $2^d = 4$ vertices are needed as proven in the following lemma.

Lemma 1. *Every polygon of VC-dimension* 2 *has at least six vertices.*

Proof. A polygon with 3 vertices is convex and thus is of VC-dimension 0. Suppose a polygon with 4 vertices has a shatterable vertex set $\{g_1, g_2\}$. Then every vertex is a neighbor of g_1 or g_2. Hence there is no vertex that sees neither g_1 nor g_2, leading to a contradiction.

Suppose a polygon with 5 vertices has a shatterable vertex set $\{g_1, g_2\}$.

Case 1: g_1 and g_2 are not neighbors. Then every vertex is a neighbor of g_1 or g_2. Hence there is no vertex that sees exactly the empty set, a contradiction.

Case 2: g_1 and g_2 are neighbors. Then both g_1 and g_2 see the set $\{g_1, g_2\}$. Therefore the remaining three vertices must cover the remaining three sets $\{g_1\}$, $\{g_2\}$ and $\emptyset$. There is exactly one vertex $v_\emptyset$ that is a neighbor of neither g_1 nor g_2, so no other vertex can see exactly the empty set.

Because the neighbor v_1 (v_2) of g_1 (g_2) sees its neighbor, it must not see g_2 (g_1), thus the internal angles must be both obtuse. Hence v_1, v_2 both have to lie 'below' the line segment g_1g_2. The fifth vertex $v_\emptyset$ is adjacent to both v_1 and v_2. It has to lie above the line segment g_1g_2 as otherwise edges of the polygon would cross (see Fig. 2). But then no vertex blocks $v_\emptyset$ from seeing g_1, which is a contradiction to $v_\emptyset$ not seeing any VC-point. □

The lemma establishes a lower bound. A polygon with VC-dimension 2 and the number of vertices matching the lower bound of 6 is depicted in Fig. 1 (middle).

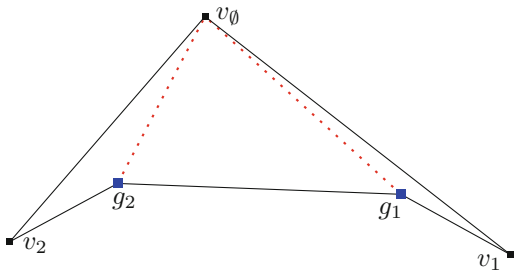


Fig. 2. A polygon with two designated VC-points that cannot be realized. The vertex $v_\emptyset$ should see neither g_1 nor g_2 but cannot be placed in such a way.

2.2 Minimum Polygons for VC-Dimension ≥ 3

For VC-dimensions 3, 4 and 5 somehow surprisingly $n_d = 2^d$ vertices suffice to construct a polygon of the respective class. In Fig. 3 we give examples of such minimum polygons. As the trivial lower bounds for the vertex count are met here, we do not require any additional proofs for the examples to be minimum representatives of their respective classes. Note that all of these polygons are also x-monotone. While the minimum polygons up to $d = 4$ were constructed manually, we generated and minimized random polygons to get the example for VC-dimension 5 (see Sect. 5.2).

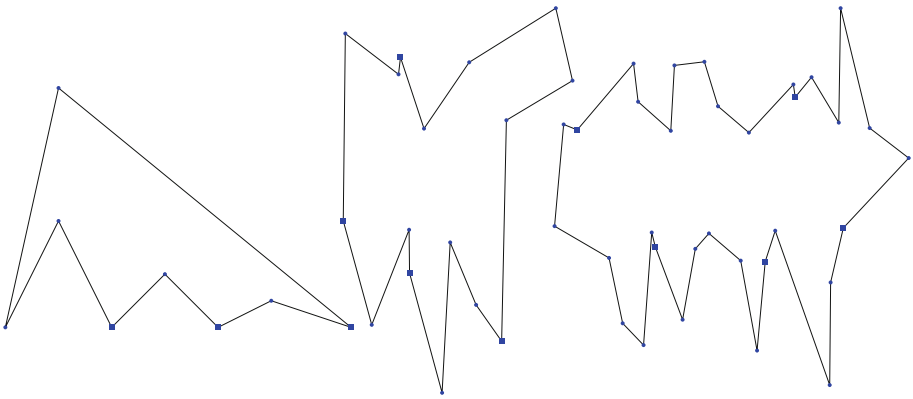


Fig. 3. Image of minimum polygons with VC-dimensions 3, 4 and 5 from left to right.

For VC-dimension 6 the smallest example we currently have consists of 78 vertices (see Fig. 4). The example is derived from a polygon with VC-dimension 6 which is essentially due to Gibson et al. [9]. Although we found the example

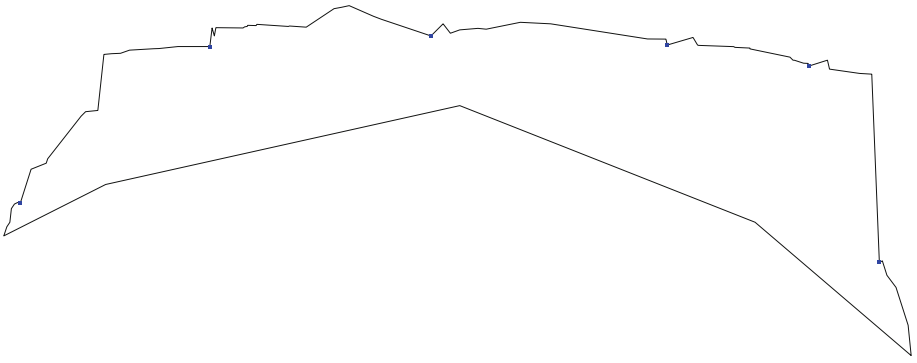


Fig. 4. Image of a polygon with VC-dimension 6.

polygon in [9] to be not of VC-dimension 6, thankfully Erik Krohn provided a corrected polygon with VC-dimension 6 with 94 vertices. We then applied minimization techniques to get the vertex count down to 78.

3 Visibility Structure

Here we introduce a notion of a *visibility string* of a polygon. The visibility string is a description of the structure of a polygon in regard to a fixed maximum shattered subset (if there are more than one). This abstraction aims at grouping together polygons that are similar to each other. For example if the position of a vertex in a polygon is changed but the pairwise visibility of viewpoints and VC-points is not affected, then the original polygon and the modified one are essentially the same for our purpose. The visibility string will also be invariant under translation, rotation, scaling and reflection of a polygon.

More precisely, for a given polygon the visibility string describes which vertices are the VC-points and for each vertex how many VC-points it can see. The visibility string consists of one symbol for each vertex in order around the polygon to form a word of length n. To account for cyclic permutation and reflection we choose a unique string:

Definition 2 (Visibility String). *Let $p_1, \ldots, p_n$ be the vertices and $g_1, \ldots, g_d$ the VC-points of a polygon in clockwise order. For a vertex p let $\mathrm{n_v}(p)$ be the number of VC-points that it sees. The symbol representing the vertex p is the $\mathrm{n_v}(p)$-th letter of the latin alphabet if p is a VC-point or else the digit $\mathrm{n_v}(p)$.*

The visibility string *is the lexicographically smallest string that can be built by concatenating the symbols of the vertices in clockwise or counterclockwise order starting at an arbitrary vertex. Here digits are smaller than letters.*

By definition every visibility string starts with a `0` symbol. Figure 5 shows an example of a visibility string.

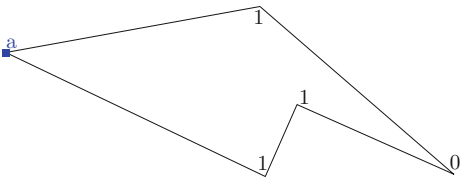


Fig. 5. A polygon with the visibility string `011a1`.

It is easy to see that the only possible visibility strings for minimum polygons of VC-dimension 0 and 1 are `000` and `01a1`, respectively. We show that there are also only a few possible visibility strings for minimum polygons of VC-dimension 2 and 3. The following proof uses the notation $\partial(p, q)$ for two vertices p, q to denote the set of all vertices that are reached when going along the border of the polygon in clockwise order starting at p and ending at q.

Theorem 1. *There are exactly four possible visibility strings for minimum polygons of VC-dimension* 2*. These strings are* `01bb12`*,* `02a2a2`*,* `01a2a2` *and* `01a2a1`*.*

Proof. Let P be a polygon with six vertices and VC-dimension 2 and $\{g_1, g_2\}$ a shattered subset where $\partial(g_1, g_2)$ contains not more vertices than $\partial(g_2, g_1)$. Consider the number c of vertices on $\partial(g_1, g_2)$, i.e. the number of vertices between g_1 and g_2. This number c cannot be 2 as otherwise every vertex apart from g_1 and g_2 itself are neighbors of g_1 or g_2. Hence no vertex would see exactly the empty set.

Consider the case $c = 1$. If g_1 and g_2 see each other, then the vertex between them lies w.l.o.g. below the line $\overline{g_1 g_2}$ (where the interior of the polygon extents above this line). Then the other three points have to see at most one VC-point. Hence the neighbors of g_1 and g_2 also have to lie below the line $\overline{g_1 g_2}$. So it is not possible that g_1 and g_2 see each other, leaving the possibility for the visibility strings `02a2a2`, `01a2a2` and `01a2a1`.

Now consider the case $c = 0$, i.e. g_1 and g_2 are neighbors. Then they both see both VC-points. One of the two vertices that is not adjacent to g_1 or g_2 has to be $v_\emptyset$. One of the neighbors of g_1 or g_2 only sees its neighboring VC-point as two of the three unspecified points see exactly one VC-point—say the neighbor of g_1. If the neighbor of g_2 sees only g_2, then we get the visibility string `01bb12` because the last unspecified point has to obscure $v_\emptyset$ and sees both VC-points. If the other neighbor sees both VC-points then there is no possibility to place the vertex v_2 that sees g_2 but not g_1. This is because neither v_1 nor $v_\emptyset$ can block it from seeing g_1, see Fig. 6. ◻

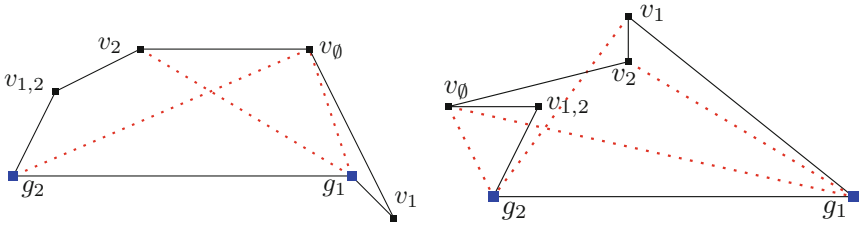


Fig. 6. Impossible visibility constraints of a polygon with two VC-points. Neither when v_2 neighbors $v_{1,2}$ (left image) nor when $v_\emptyset$ neighbors $v_{1,2}$ (right image) is there a vertex that blocks v_2 from seeing g_1.

For VC-dimension 2 there is some leeway because there are more vertices than subsets to cover but this does not hold for VC-dimension 3.

Theorem 2. *There are exactly two possible visibility strings for minimum polygons of VC-dimension 3. These strings are* `02a2a2a3` *and* `02a2a3a2`*.*

Interestingly for VC-dimension 4 the number of possible visibility strings increases dramatically: In 800 analyzed minimum polygons, we identified about

600 different visibility strings. Nevertheless, a relaxed definition of the visibility string which considers only the configuration of the VC-points and their visibility with respect to each other could provide better structural insights: In about 71,000 randomly generated polygons only two possible configurations were found.

4 Computation of the Visibility VC-Dimension

In this section we discuss the methods for computing the VC-dimension of a given polygon. This allows to determine better instance-based approximation guarantees for polygons with $d < 6$ when applying (art gallery) algorithms with a VC-dimension dependent approximation guarantee. Furthermore, we leverage tools for efficient VC-dimension computation later to analyze the distribution of the VC-dimension classes among sets of randomly generated polygons.

4.1 Naive Algorithm and Improvements

Given an input polygon with n vertices, we want to find the largest shattered subset. As we already know that no subset of size 7 or larger can be shattered, it suffices to check all subsets up to size 6. If the subsets are considered in increasing order of their size, the process can be stopped if no subset of a certain size k is shattered (as then naturally no superset can be shattered). Hence, for a polygon with VC-dimension d we have to check $\mathcal{O}(\min(n^{d+1}, n^6))$ subsets.

To determine whether a certain subset S of size $k \leq 6$ is shattered, we need to check whether there are 2^k suitable viewpoints (each being visible from one particular subset of S). Within a simple polygon, the so called visibility region of a point can be computed in linear time. Hence it takes $\mathcal{O}(nk) = \mathcal{O}(n)$ time to retrieve the visibility regions of all VC-points. From those we can interfere in linear time if there is a viewpoint for each subset. Hence the total running time amounts to $\mathcal{O}(\min(n^{d+2}, n^7))$. Note that in practice it makes sense to have an initial phase where the visibility region of each vertex is precomputed to avoid redundant computations.

The algorithm can be further improved by skipping unnecessary tests for certain vertex sets: If the algorithm finds a shattered set S' of size k, it can omit testing other sets of the same cardinality. In addition, if the algorithm considers sets of the same cardinality in lexicographic order, many sets can be skipped when the algorithm starts to consider the sets with cardinality $k+1$. In fact, all sets with a lexicographically smaller prefix than S' do not need to be checked as already their prefixes could not be shattered. For example, if the set $\{2, 4\}$ is the first of size 2 to be shattered, the algorithm skips all remaining sets with 2 vertices. And it also skips the sets $\{0, 1, 2\}, \{0, 1, 3\}, \ldots, \{2, 3, n-2\}, \{2, 3, n-1\}$ because the sets $\{0, 1\}, \ldots, \{2, 3\}$ could not be shattered. Instead it proceeds with the set $\{2, 4, 5\}$. While these rules have the potential to reduce the numbers of sets to be checked significantly, they unfortunately can not circumvent that for $d < 6$ all sets of size $d + 1$ need to be checked in order to certify that no subset larger than d is shattered.

4.2 Stack-Based Algorithm

To reduce the number of large sets that have to be tested, we propose a stack-based approach. Here, we assure that a set is only tested if already all of its prefixes were shown to be shattered.

Let S be an initially empty stack and $v := 0$. Repeat the following steps until the algorithm terminates: Test whether $S \cup \{v\}$ is shattered. If this is the case, then push v onto the stack. Set $v := v + 1$. If $v = n - 1$ and the stack is not empty, then set $v := S.pop() + 1$. If $v = n - 1$ and the stack is empty, then terminate and return (the size of) the biggest found shattered set.

5 Experiments

We implemented the naive algorithm for VC-dimension computation along with the described improvements, and our stack-based algorithm. The used programming language is Rust. Experiments were run on a machine with an Intel i5 CPU with a frequency of 2.5 GHz and 8 GB of main memory. Note that the size of the memory was no limitation as the algorithms all used at most a few MB.

5.1 VC-Dimension Computation

First of all, we investigated how many vertex subsets of a randomly generated polygon are tested in the described approaches for VC-dimension computation. Results for polygons with VC-dimension 4 are shown in Fig. 7. We observe that the stack-based method clearly outperforms the naive method, even if we look at the maximum number of sets tested for the former. The ratio gets drastically lower the larger the number of vertices, so the advantage of the stack-based method clearly increases with the size of the polygon. Further experiments also showed that the ratio gets lower the larger the VC-dimension of the polygon is. For a clock time comparison we computed the VC-dimension of 10 polygons with 100 vertices each and $d = 4$. The naive algorithm took nearly 13 min in total whereas the stack based algorithm only took 5 s.

5.2 VC-Dimension Distribution in Random Polygons

In order to find minimum polygons and analyze related algorithms, we generated large sets of random polygons. Initially, we used two classic methods for random polygon generation: 2-Opt moves and Quick Star [10]. For $100,000$ polygons created for different vertex counts, we computed d. The resulting distributions are given in Fig. 9, top and middle. We observe that the vast majority of polygons generated with these methods exhibit a VC-dimension of 3 or smaller. While Quick Star leads to significantly more polygons with $d = 4$ than 2-Opt, even in rather large polygons with 50 vertices still about half of them only exhibit a VC-dimension of 3. The reason why 2-Opt primarily produces polygons of low VC-dimension is that there often are multiple areas that are mostly independent

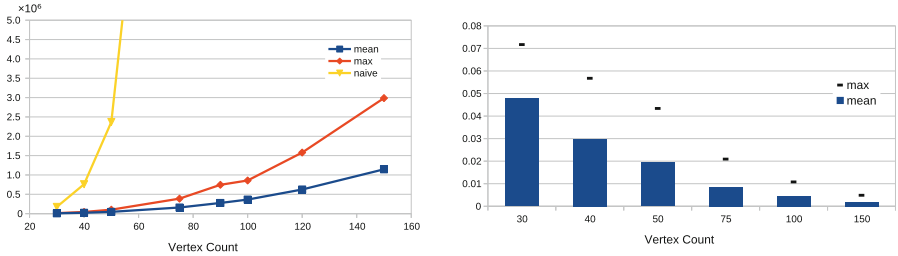


Fig. 7. Number of sets considered using the stack-based method (mean and max) versus the naive method for polygons with VC-dimension 4. To the left are the absolute numbers, to the right the ratios.

of each other in the sense that most vertices of one area do not see any one from the other areas. Quick Star polygons exhibit a more compact shape and more likely contain viewpoints with large visibility regions. Figure 8, left and middle, shows examples of polygons created with 2-Opt and Quick Star.

To get more example polygons of higher VC-dimension, we developed another method for generating (x-monotone) polygons randomly: First, the plane is divided into three horizontal regions *top*, *bottom* and *gap*. Then, $(n-2)/2$ points are placed randomly into the top region and just as many in the bottom region. One point is put in the horizontal gap between the top and bottom regions as the leftmost point and one as the rightmost point. These points are connected in the following order to always get an x-monotone polygon: Left point, bottom points in ascending x-direction, right point, top points in descending x-direction. We refer to this method as X-Gen. An example of a polygon created with X-Gen is shown in Fig. 8, right.

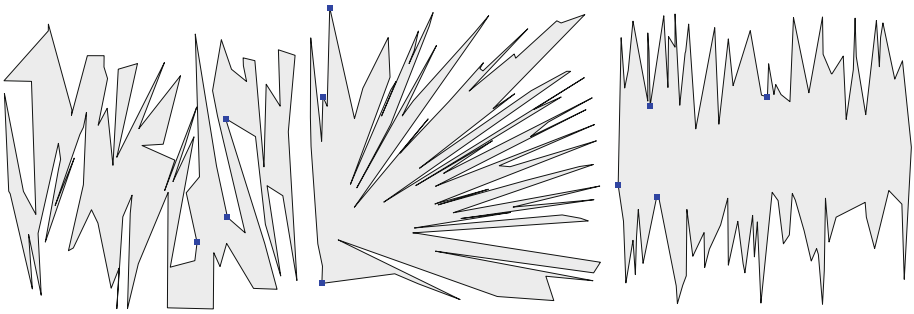


Fig. 8. Examples of randomly generated polygons with 100 vertices each. The methods used are from left to right: 2-Opt, Quick Star, X-Gen.

Figure 9, bottom, shows the respective distribution of the VC-dimension in polygons generated via X-Gen. We observe that for polygons consisting of 40 or more vertices almost all generated examples have a VC-dimension of 4.

Fig. 9. Distribution of the VC-dimension on randomly generated polygons with a given number of vertices. The polygons were generated from top to bottom with the 2-Opt method, Quick Star and X-Gen.

The 2-Opt method never produced a polygon with VC-dimension 5 even for larger vertex counts. Generating 100,000 polygons with 150 vertices, Quick Star produced 17 polygons with VC-dimension 5 and X-Gen 49. Using X-Gen we also found an example with $d = 5$ that could be minimized to consist of only 32

vertices (by iterative vertex removal), therefore producing a minimum member of the respective VC-dimension class.

6 Conclusion and Open Problems

In this paper we studied minimum polygons of fixed VC-dimension d. For $d \leq 5$, we identified example polygons of minimum size. For $d = 6$, the size of the minimum polygon in this class remains an open question. Based on the results for $d = 3, 4, 5$ and our current example of a polygon with 78 vertices, we conjecture that a polygon with 64 vertices and VC-dimension 6 might exist. Better structural insights, for example by extending the notion of visibility strings as defined in this paper, might help to construct such an example.

For computing the VC-dimension of a given polygon, we devised an algorithm which is sufficiently fast in practice to compute the VC-dimension of polygons with a few hundred vertices. The theoretical running time is $\mathcal{O}(\min(n^{d+2}, n^7))$ and hence output-sensitive. It might however be possible to compute the VC-dimension significantly faster (and independent of d); for example by considering the intersections of the visibility regions of all vertices.

Finally, it is worthwhile to investigate further polygon generating methods which are capable of producing either polygons with a given VC-dimension or at least lead to a larger percentage of polygons with high VC-dimension. Our devised X-Gen method already produces more complex polygons on average (in terms of the visibility structure) than conventional methods as 2-Opt or Quick Star, but other custom-tailored methods might perform even better. The resulting polygons could then form interesting benchmark sets for algorithms and heuristics that solve art gallery problems.

References

1. Eidenbenz, S.: Inapproximability results for guarding polygons without holes. In: Chwa, K.-Y., Ibarra, O.H. (eds.) ISAAC 1998. LNCS, vol. 1533, pp. 427–437. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-49381-6_45
2. Eidenbenz, S., Stamm, C., Widmayer, P.: Inapproximability results for guarding polygons and terrains. Algorithmica **31**(1), 79–113 (2001)
3. Ghosh, S.K.: Approximation algorithms for art gallery problems. In: The Proceedings of Canadian Information Processing Society Congress, pp. 429–434 (2010)
4. Valtr, P.: Guarding galleries where no point sees a small area. Israel J. Math. **104**(1), 1–16 (1998)
5. Brönnimann, H., Goodrich, M.T.: Almost optimal set covers in finite VC-dimension. Discrete Comput. Geom. **14**(4), 463–479 (1995)
6. Gibson, M., Krohn, E., Wang, Q.: The VC-dimension of visibility on the boundary of a simple polygon. In: Elbassioni, K., Makino, K. (eds.) ISAAC 2015. LNCS, vol. 9472, pp. 541–551. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48971-0_46
7. Gilbers, A., Klein, R.: A new upper bound for the VC-dimension of visibility regions. In: Proceedings of the Twenty-seventh Annual Symposium on Computational Geometry, SoCG 2011, New York, NY, USA, pp. 380–386. ACM (2011)

8. Gilbers, A.: VC-dimension of perimeter visibility domains. Inf. Process. Lett. **114**(12), 696–699 (2014)
9. Gibson, M., Krohn, E., Wang, Q.: On the VC-dimension of visibility in monotone polygons. In: Canadian Conference on Computational Geometry, pp. 85–94 (2014)
10. Auer, T., Held, M.: Heuristics for the generation of random polygons. In: Proceedings of the 8th Canadian Conference on Computational Geometry, pp. 38–43. Carleton University Press (1996)

Minsum k-Sink Problem on Dynamic Flow Path Networks

Robert Benkoczi[1], Binay Bhattacharya[2], Yuya Higashikawa[3], Tsunehiko Kameda[2(✉)], and Naoki Katoh[4]

[1] Department of Mathematics and Computer Science, University of Lethbridge, Lethbridge, Canada
[2] School of Computing Science, Simon Fraser University, Burnaby, Canada
tiko@sfu.ca
[3] School of Business Administration, University of Hyogo, Kobe, Japan
[4] School of Science and Technology, Kwansei Gakuin University, Sanda, Japan

Abstract. In emergencies such as earthquakes, nuclear accidents, etc., we need an evacuation plan. We model a street, a building corridor, etc. by a path network, and consider the problem of locating a set of k sinks on a dynamic flow path network with n vertices, where people are located, that minimizes the sum of the evacuation times of all evacuees. Our minsum model is more difficult to deal with than the minmax model, because the cost function is not monotone along the path. We present an $O(kn^2 \log^2 n)$ time algorithm for solving this problem, which is the first polynomial time result. If the edge capacities are uniform, we give an $O(kn \log^3 n)$ time algorithm.

1 Introduction

Due to many recent disasters such as earthquakes, volcanic eruptions, hurricanes, and nuclear plant accidents, evacuation planning is getting increasing attention. The evacuation k-sink problem is an attempt to model evacuation in such emergency situations [5,6]. In this paper, a *k-sink* means a set of k sinks that minimizes the sum of the evacuation time of every evacuee to a sink. Researchers have worked mainly on two objective functions. One is the evacuation completion time (minmax criterion), and the other is the sum of the evacuation times of all the evacuees (minsum criterion). It is assumed that all evacuees from a vertex evacuate to the same sink.

Mamada et al. [12] solved the minmax 1-sink problem for dynamic flow tree networks in $O(n \log^2 n)$ time under the condition that only a vertex can be a sink. When edge capacities are uniform, Higashikawa et al. [9] and Bhattacharya and Kameda [3] presented $O(n \log n)$ time algorithms with a more relaxed

This work was supported in part by NSERC Discovery Grants, awarded to R. Benkoczi and B. Bhattacharya, in part by JST CREST Grant Number JPMJCR1402 held by N. Katoh and Y. Higashikawa, and in part by JSPS Kakenhi Grant-in-Aid for Young Scientists (B) (17K12641) given to Y. Higashikawa.

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 78–89, 2018.
https://doi.org/10.1007/978-3-319-94667-2_7

condition that the sink can be on an edge. Chen and Golin [4] solved the minmax k-sink problem on dynamic flow tree networks in $O(k^2 n \log^5 n)$ time when the edge capacities are non-uniform. Regarding the minmax k-sink on dynamic flow path networks, Higashikawa et al. [10] present an algorithm to compute a k-sink in $O(kn)$ time if the edge capacities are uniform. In the general edge capacity case, Arumugam et al. [1] showed that a k-sink can be found in $O(kn \log^2 n)$ time. Bhattacharya et al. [2] recently improved these results to $O(\min\{n \log n, n + k^2 \log^2 n)$ time in the uniform edge capacity case, and to $O(\min\{n \log^3 n, n \log n + k^2 \log^4 n\})$ time in the general case.

The minsum objective function for the sink problems is motivated, among others, by the desire to minimize the transportation cost of evacuation or the total amount of psychological duress suffered by the evacuees. It is more difficult than the minmax variety because the objective cost function is not unimodal, and, to the best of our knowledge, practically nothing is known about this problem on more general networks than path networks. A path network, although simple, can model an airplane aisle, a hall way in a building, a street, a highway, etc., to name a few. For the simplest case of $k = 1$ and uniform edge capacities, Higashikawa et al. [10] proposed an $O(n)$ time algorithm. For the case of general k and uniform edge capacities, Higashikawa et al. [10] showed that a k-sink can be found in time bounded by $O(kn^2)$ and $2^{O(\sqrt{\log k \log\log n})} n^2$.

The main contribution of this paper is an $O(kn^2 \log^2 n)$ time algorithm, which is achieved by a novel data structure and the concepts of *cluster* and *section* in an evacuee flow. Our second algorithm solves the problem in $O(kn \log^3 n)$ time if the edge capacities are the same.

This paper is organized as follows. In the next section, we define some terms that are used throughout this paper, and present a few basic facts. Section 3 formulates the framework for solving the minsum k-sink problem, utilizing Dynamic Programming (DP), and provides a solution. In Sect. 4, we introduce the concepts of cluster and section which play a key role in subsequent discussions, and discuss how to compute the local cost data that are required in our DP formulation. Section 5 states our main theorem, which results from the preceding section. Finally, Sect. 6 concludes the paper.

2 Preliminaries

Let $P(V, E)$ denote a given path network, where the vertex set V consists of $v_1, v_2, \ldots, v_n$, which we assume to be arranged in this order, from left to right horizontally.[1] Vertex v_i has weight $w_i \in \mathbb{Z}_+$, representing the number of evacuees initially located at v_i, and edge $e_i = (v_i, v_{i+1}) \in E$ has *length* or *distance* d_i (> 0) and *capacity* c_i, which is the upper limit on the flow rate through e_i in persons/unit time. We write $v_i \prec v_j$ if $i < j$. For two vertices $v_i \prec v_j$, the subpath between them is denoted by $P[v_i, v_j]$, and $d(v_i, v_j)$ (resp. $c(v_i, v_j)$) denotes

[1] In Sects. 3 and 4, for simplicity, we will often identify a vertex with its index, referring to vertex i, instead of vertex v_i.

its length (resp. the minimum capacity of the edges on $P[v_i, v_j]$). It takes each evacuee τ units of time to travel a unit distance.

Our model assumes that the evacuees at all the vertices start evacuation at the same time at the rate limited by the capacity of the outgoing edge. It also assumes that all the evacuees at a non-sink vertex who were initially there or who arrive there later evacuate in the same direction (either to the left or to the right), i.e., the evacuee flow is *confluent*. We sometimes use the term "cost" to refer to the aggregate evacuation time of a group of evacuees to a certain destination. A *k-sink* shares the following property of the *median* problem [11].

Lemma 1 [10]**.** *There is a k-sink such that all the k sinks are at vertices.*

If we plot the arrival flow rate at, or departure flow rate from, a vertex as a function of time, it consists of a sequence of *(temporal) clusters*. The *duration* of a cluster is the length of time in which the flow rate corresponding to the cluster is greater than zero. A cluster consists of a sequence of *sections*, such that any adjacent pair of sections have different heights. In other words, a section is a maximal part of a cluster with the same *height* (= flow rate). A *simple cluster* consists of just one section. We say that a cluster/section *carries* (the evacuees on) a vertex, if those evacuees provide flow to the cluster/section. A time interval of flow rate 0 between adjacent clusters is called a *gap*. These terms are illustrated in Fig. 1. Unless otherwise specified, we assume that evacuees arrive at vertex v_i from vertex v_{i+1}. The case where the evacuees move rightward can be treated symmetrically.

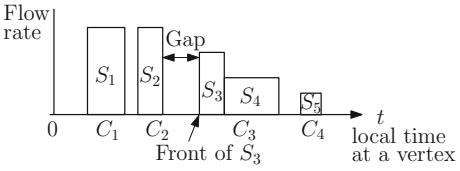


Fig. 1. Terms used: $\{S_i\}$ are sections and $\{C_j\}$ are clusters.

The *front* of a cluster/section is the time when it starts. The *first vertex* of a cluster is the vertex from which the evacuee corresponding to the front of the cluster originates. The *offset* of a cluster with respect to vertex v_i is the time until the first evacuee belonging to the cluster arrives at v_i. For $v_j \prec x \prec v_{j+1}$, we define the following costs.

$$\Phi_{L,j}(x) \triangleq \text{cost contribution to } x \text{ from } P[v_1, v_j],$$
$$\Phi_{R,j}(x) \triangleq \text{cost contribution to } x \text{ from } P[v_{j+1}, v_n],$$
$$\Phi(x) = \begin{cases} \Phi_{L,j}(x) + \Phi_{R,j+1}(x) & \text{if } v_j \prec x \prec v_{j+1} \\ \Phi_{L,j-1}(x) + \Phi_{R,j+1}(x) & \text{if } x = v_j. \end{cases} \quad (1)$$

A point $x = \mu$ that minimizes $\Phi(x)$ is called a *minsum 1-sink*.

The total cost is the sum of the costs of all sections. The cost of a section of height c with offset t_0 and duration δ_t is given by

$$\lambda t_0 + \frac{\lambda^2}{2c}, \tag{2}$$

where $\lambda = c\delta_t$ is the number of evacuees carried by the section [8]. To be exact, the ceiling function must be applied to the second term in (2), but we omit it for simplicity, and *adopt* (2) as our objective function [5]. Or we can consider each molecule of a fluid-like material as an "evacuee." The average evacuation time for an evacuee carried by this section is $t_0 + \lambda/2c$, where $\lambda/2c$ represents the average delay to reach the front vertex of the section, and the aggregate is given by $(t_0 + \lambda/2c) \times \lambda$, which yields (2). We call the first (resp. second) term in (2) the *extra cost* (resp. *intra cost*) of the section. A *minsum k-sink* partitions the path into k subpaths, and places a 1-sink on each subpath in such a way that the sum of the evacuation time of every evacuee to a sink is minimized.

3 DP Formulation

We first present a dynamic programming (DP) formulation that follows the template of recursive functions proposed by Hassin and Tamir [7] for the p-median problem. Our innovation consists in the manner in which we process the recursive computations efficiently, given that the cost functions for the sink location problem are significantly more difficult to compute than those for the regular median problem. Our algorithm is more general in that it relies only on one fundamental property of some cost functions, i.e., monotonicity.

3.1 Derivation of Recurrence Formulae

Let $F^k(i)$, $1 \le k \le i \le n$, denote the minsum cost when k sinks are placed on subpath $P[v_1, v_i]$. Similarly, define $G^k(i)$, $1 \le k \le i \le n$, as the minsum cost when k sinks are placed on subpath $P[v_1, v_i]$, *and* v_i is the rightmost sink. We start with $i = k+1$, since $F^k(i) = G^k(i) = 0$ for $i \le k$. For $j < i$, we also define $R(j, i)$, which is the cost of evacuating all the evacuees on subpath $P[v_{j+1}, v_i]$ to v_j, and $L(j, i)$, which is the cost of evacuating all the evacuees on subpath $P[v_j, v_{i-1}]$ to v_i. By definition, we have

$$F^k(i) = \min_{k \le j \le i} \{G^k(j) + R(j, i)\}, \tag{3}$$

$$G^k(i) = \min_{k \le j \le i} \{F^{k-1}(j) + L(j+1, i)\}. \tag{4}$$

To solve the above recursive equations, we clearly need to compute functions $R(j, i)$ and $L(j, i)$. Moreover, to obtain a DP algorithm with time complexity sub-quadratic in n, we also need to quickly find the index j that minimizes the recurrence relations (3) and (4). Note that to get $F^k(i)$, we need to compute $\{G^p(\cdot), F^p(\cdot)\}$ for $p = 1, 2, \ldots, k$.

To motivate our approach, let us plot points $(G^k(j), R(j,i))$ in a 2-dimensional coordinate system for all j, $1 \leq j \leq i$, for a fixed vertex v_i. See Fig. 2. If we superimpose $G^k(j) + R(j,i) = c$ for a given value c in the same coordinate system, it is a $-45°$ line. If we increase c from 0, this line eventually touches one of the plotted points. The first point it touches gives the optimal value that minimizes $G^k(j) + R(j,i)$. In Fig. 2, this optimal is given by the point $(G^k(j_1), R(j_1,i))$. For convenience, let us refer to point $(G^k(j), R(j,i))$ as *point* (j,i).

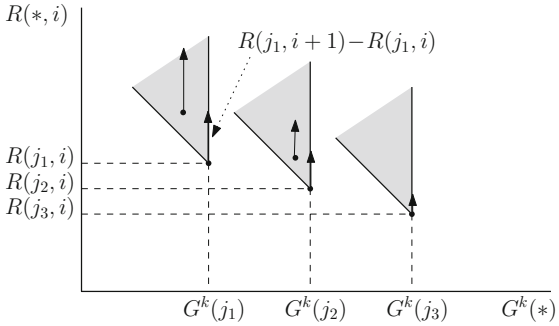


Fig. 2. $R(*,i)$ vs. $G^k(*)$. $j_1 < j_2 < j_3$.

We now explain that this representation provides us very useful information. To see it, for each point (j,i), define the *V-area* that lies above the $-45°$ line and to the left of the vertical line through it as shown as a shaded area in Fig. 2. We say that a point (j,i) situated in the V-area of another point (j_s,i) is *dominated* by (j_s,i), since the cost of point (j,i) is higher than the cost of (j_s,i). We sometimes say that v_j is dominated by v_{j_s}, when i is clear from the context. Thus the points at the bottoms of the V-areas are the only non-dominated points. For subpath $P[v_1,v_i]$ let $J(i) = \{j_1,\ldots,j_{g(i)}\}$, where $j_1 \leq j_2 \leq \ldots \leq j_{g(i)} \leq i$ and $\{(j_s,i) \mid s = 1,\ldots,g(i)\}$, are the set of all points at the bottoms of the V-areas. From the above discussion the following lemma follows directly.

Lemma 2. *$F^k(i) = G^k(j_1) + R(j_1,i)$ would hold if the path ended at vertex v_i.*

Function $G^k(i)$ can be computed in a similar manner. Let us now compare $J(i+1)$ for $P[v_1,v_{i+1}]$ with $J(i)$ for $P[v_1,v_i]$. Since $j_{g(i)} \leq i$, vertex v_{i+1} is farther from v_{j_s} than it is from v_{j_t}, if $s < t$. We thus have

$$R(j_s,i+1) - R(j_s,i) \geq R(j_t,i+1) - R(j_t,i) \quad \text{for } s < t. \tag{5}$$

The arrows in Fig. 2 indicate the increase $R(*,i+1) - R(*,i)$ in computing $J(i+1)$, compared with $J(i)$. Moreover, if (j,i) is dominated by (j_s,i), then point (j,i') will also be dominated by (j_s,i') for any $i' > i$. This implies that once it is determined that $(j,i) \notin J(i)$, then (j,i') will not belong to $J(i')$ for any $i' > i$. We will discuss how to update $J(i)$ to $J(i+1)$ in the next subsection.

3.2 Computing Switching Points

We compute $F^k(i)$ by Lemma 2, maintaining the set $J(i)$ of non-dominated candidate vertices. However, not all vertices in $J(i)$ that are non-dominated when computing $F^k(i)$ remain useful because some of these vertices may become dominated when computing $F^k(i')$ for $i' > i$. We shall identify those vertices, as they become dominated, and discard them.

Let us denote by $x(j_{s-1}, j_s)$, $1 < s \leq g(i)$, the *switching point*, namely the leftmost vertex $v_{i'}$ $(g(i) < i' \leq n)$, if any, for which j_s dominates j_{s-1}. If such an index does not exist, it means that j_s never dominates j_{s-1} and therefore we need not remember j_s. For convenience, let us introduce *a dummy vertex* j_0 so that we can write $x(j_0, j_1) = j_1$. Formally, we have

$$x(j_{s-1}, j_s) = \begin{cases} \min\{i' : [j_s < i' \leq n] \wedge [G^k(j_s) + R(j_s, i') \\ \qquad\qquad \leq G^k(j_{s-1}) + R(j_{s-1}, i')]\} & \text{if } s \geq 2, \\ j_s & \text{if } s = 1. \end{cases} \tag{6}$$

Computing and maintaining the sequence $x(j_0, j_1), \ldots, x(j_{g(i)-1}, j_{g(i)})$ allows us to determine a subset of non-dominated vertices v_{j_s}, which are potentially optimal vertices that may minimize function $F^k(\cdot)$ later. We therefore assume $x(j_0, j_1) < x(j_1, j_2) < \ldots < x(j_{g(i)-1}, j_{g(i)})$. To see this, assume for example that $x(j_2, j_3) < x(j_1, j_2)$ holds. Then j_2 will never be an optimal vertex, because for large enough i $(\geq x(j_1, j_2))$ which makes v_{j_2} dominate v_{j_1}, vertex v_{j_3} already dominates v_{j_2}, since $x(j_2, j_3) < x(j_1, j_2)$. This implies that j_2 can be removed from $J(i)$.

Updating set $J(i) = \{j_1, \ldots, j_{g(i)}\}$ to $J(i+1)$.

Change 1: If v_{j_1} becomes dominated by v_{j_2} (i.e., if $x(j_1, j_2) = i + 1$), then remove j_1 in constructing $J(i+1)$. (Note that $x(j_2, j_3) > i + 1$.)

Change 2: Starting from the last vertex in $J(i)$, find the rightmost vertex v_{j_s}, if any, that is not dominated by v_{i+1}. If none, let $s = 0$. Remove j_{s+1} to $j_{g(i)}$ in $J(i)$ to obtain $J(i+1)$. Put v_{i+1} in $J(i+1)$ as the last vertex $j_{g(i+1)}$.

It is easy to show that computing $J(i+1)$ from $J(i)$ takes amortized $O(t_X(n))$ time, where $t_X(n)$ is the time needed to compute $x(j, j')$ value for one arbitrary pair (j, j'), $j < j'$.

Based on the above discussion, we present Algorithm 1 below that shows a skeleton of our method for computing a minsum k-sink. In it, variable $J = \{j_1, j_2, \ldots, j_{g(i)}\}$ represents the ordered set of candidate vertices which is updated from iteration to iteration, and $X = \{x(j_0, j_1), x(j_1, j_2), \ldots, x(j_{g(i)-1}, j_{g(i)})\}$ represents the corresponding ordered set of switching points.

Lemma 3. *The minsum k-sink in dynamic flow path networks can be found in $O(kn \cdot t_X(n))$ plus preprocessing time.*

Algorithm 1. Minsum k-sink algorithm

1 **Input Data:** Number of evacuees w_i at each vertex i; Capacity c_i and length d_i for each edge e_i; An integer k representing the number of sinks to be located; $\mathcal{T}$ and $\mathcal{C}$ (defined in Sect. 4);
2 **Outputs:** A set $S^* \subseteq \{1, \ldots, n\}$ of k sinks to be located; Cost Z^* of solution S^*;
3 Base case: compute $G^1(i)$ for all $i \in \{1, \ldots, n\}$;
4 **for** $p \in \{1, \ldots, k\}$ **do**
5 $J \leftarrow 1; X \leftarrow 1;$ `//Initialize candidate sequence` J `and` $x(\emptyset, 1) = 1$
6 $F^p(1) \leftarrow G^p(1);$ `//` $L(1,1) = 0$
7 **for** $i \in \{2, \ldots, n\}$ **do**
8 **repeat** `//Update the candidate list` J `by considering` i
9 **if** J *is empty* **then**
10 $J \leftarrow i; X \leftarrow i;$ `//New vertex` v_i `is the dominating vertex`
11 *done* $\leftarrow$ `true`
12 **else**
13 $j \leftarrow$ last item in J; $x \leftarrow$ last item in X;
14 Compute $x(j, i)$; `//Switching point between` i `and` j
15 **if** $x(j, i)$ *does not exist* **then**
16 *done* $\leftarrow$ `true`; `//` i `is dominated`
17 **else if** $x(j, i) > x$ **then** `//` i `does not dominate` j
18 Append i at the end of J; append $x(j, i)$ at the end of X;
19 *done* $\leftarrow$ `true`
20 **else** `//` i `dominates` j
21 Remove j from the end of J; remove x from the end of X;
22 *done* $\leftarrow$ `false`
23 **end**
24 **end**
25 **until** *done*;
`// Check for Change 1 (in Sect. 3.2) and compute` $F^p(i)$
26 Let x^* be the rightmost vertex in X satisfying $x^* \leq i$ and let j^* be its corresponding vertex in J;
27 Let $F^p(i) \leftarrow G^p(j^*) + R(j^*, i)$
28 **end**
29 **if** $p < k$ **then**
30 Compute $G^{p+1}(i)$ in a similar way using $F^p(i)$ for all $1 \leq i \leq n$
31 **else**
32 **return** $Z^* = F^k(n)$; `//Sink set` S^* `can be obtained from` Z^* `in a standard way`
33 **end**
34 **end**

Proof. Algorithm 1 performs $O(kn)$ iterations in lines 4 and 7. The repeat loop at line 8 executes at most as many times as the size of list J. However, an element is added to list J at most once for each iteration i (lines 10 and 18), so J cannot receive more than n elements throughout the duration of the algorithm, and the repeat loop cannot have more than n iterations throughout the duration of the

algorithm. In each iteration of the repeat loop, $x(\cdot,\cdot)$ is computed a constant number of times, and the lemma follows. □

Now that we have the above lemma, the rest of this paper is devoted to making $t_x(n)$ as small as possible, culminating in Lemmas 7 and 8.

4 Data Structures for Computing $R(j,i)$ and $L(j,i)$

Costs $R(j,i)$ and $L(j,i)$ are used in (3), (4), and (6). It is needed wherever $x(j,i)$ is used in Algorithm 1, including Line 27. We only discuss how to compute $R(j,i)$, since $L(j,i)$ can be computed similarly. To compute $R(j,i)$, we need to know the section sequence of the vertices on $P[v_{j+1}, v_i]$ arriving at v_j. To find it efficiently, during preprocessing we construct a balanced binary tree, named *cluster tree* $\mathcal{T}$, whose leaves are the vertices of P, arranged from v_1 to v_n. We also construct a capacity tree $\mathcal{C}$, which is a standard binary search tree from which one can find capacity $c(v_j, v_h)$ in $O(\log n)$ time for any pair of vertices $v_j \preceq v_h$.

For each non-leaf node[2] u of $\mathcal{T}$, let $v_L(u)$ (resp. $v_R(u)$) denote the leftmost (resp. rightmost) vertex of P that belongs to subtree $\mathcal{T}(u)$. We say that u *spans* subpath $P[v_L(u), v_R(u)]$. For a node u of $\mathcal{T}$, let $\alpha_R^u(v_j)$ (resp. $\beta_R^u(v_j)$) denote the arrival (resp. departure) section sequence at (resp. from) v_j ($\preceq v_L(u)$), carrying the vertices spanned by u. At each node u of $\mathcal{T}$, we precompute and store $\alpha_R^u(v_L(u))$ and $\beta_R^u(v_L(u))$, as we describe below.

At a leaf node, which is a vertex v_i, it is easy to construct $\beta_R^{v_i}(v_i)$, which is just a section of height c_{i-1} and duration w_i/c_{i-1} that starts at time 0 (local time at v_i). We obviously have $\alpha_R^{v_i}(v_i) = \emptyset$. For an internal node u of $\mathcal{T}$ with two child nodes, u_a and u_b, assuming that we have computed $\alpha_R^{u_a}(v_L(u_a))$, $\beta_R^{u_a}(v_L(u_a))$, $\alpha_R^{u_b}(v_L(u_b))$, and $\beta_R^{u_b}(v_L(u_b))$, we want to compute $\alpha_R^u(v_L(u))$ and $\beta_R^u(v_L(u))$ from them. Let $v_{j+1} = v_L(u_a)$. Then $\beta_R^{u_b}(v_L(u_b))$ would become $\alpha_R^{u_b}(v_{j+1})$ with a delay of $d(v_{j+1}, v_L(u_b))\tau$ according to the local time at v_{j+1}, provided it encountered no congestion on its way. If the height of an arriving section in $\alpha_R^{u_b}(v_{j+1})$ is larger than c_j, the evacuees carried by that section cannot depart from v_{j+1} at the arrival rate. See Fig. 3(a), where $S_1, S_2, \ldots$, are the sections of $\alpha_R^{u_b}(v_{j+1})$, arriving at v_{j+1}. The durations of some sections get stretched in this case, by the *ceiling* operation [12]. The following two situations can arise to these sections, when they are converted into the departing sections of $\beta_R^u(v_{j+1})$. (When S_1 arrives, there may be w (> 0) leftover evacuees at the vertex. We will consider such a scenario shortly.)

(a) A stretched section by a ceiling operation ends in a gap. (Fig. 3(b) shows that the stretched S_1 fills the next gap entirely, merges with S_2, and the following gap is partially filled. The amount equal to the light-gray parts consisting of later arrivals moves to the dark-gray parts, to fill gaps.)
(b) A section may shrink due to the expanded section preceding it, with its front pushed to a later time. (In Fig. 3(c), the stretched S_3 "swallows" a part of

[2] We use the term "node" for $\mathcal{T}$ to distinguish them from the vertices of P.

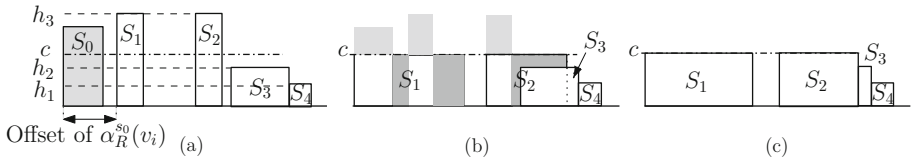


Fig. 3. (a) $\alpha_R^{u_b}(v_{j+1})$; (b) Amount equal to the light-gray parts fill the dark-gray parts; (c) Result.

S_4 and S_4 shrinks. The next section (such as S_5 in this example), if any, undergoes no change, since its height is less than c_j.)

From observations (a) and (b) above, we can easily infer the following lemma.

Lemma 4. *The heights of the sections in $\alpha_R^u(v_{j+1})$ and $\beta_R^u(v_{j+1})$ are non-increasing with time.*

Redefine $S_1, S_2, \ldots,$ to be the sections of $\alpha_R^u(v_L(u))$. To find which sections merge with other sections when a smaller capacity is encountered, we place the weight-time ratios

$$\{\lambda(S_h)/\delta_h \mid S_h \text{ is a section of } \alpha_R^u(v_L(u))\}$$

in a max-heap $\mathcal{H}_u$, where $\lambda(S_h)$ is the sum of the weights of the vertices carried by S_h, and δ_h is the time difference between the fronts of S_h and S_{h+1}. Thus in converting $\alpha_R^u(v_{j+1})$ into $\beta_R^u(v_{j+1})$, we pop out of $\mathcal{H}_u$ those ratios that are larger than c_j, and for each such ratio, we merge the corresponding pair of sections. In Fig. 3, for example, $\lambda(S_3)/\delta_3 > \lambda(S_1)/\delta_1 > c_j$, so that S_3 merges with S_4, and S_1 merges with S_2. For the resulting new sections, we compute the weight-time ratios, and if they are larger than c_j, we repeat the merging process.

We call two nodes u_a and u_b of $\mathcal{T}$ *adjacent* if $v_R(u_a)$ and $v_L(u_b)$ are adjacent vertices on P. It is easy to observe

Proposition 1. *Let u_a and u_b be two adjacent nodes of $\mathcal{T}$. The evacuees still left at vertex $v_L(u_a)$, if any, when the first evacuee in $\alpha_R^{u_b}(v_{j+1})$ (shifted $\beta_R^{u_b}(v_L(u_b))$) arrives there belong to the last cluster in $\beta_R^{u_a}(v_L(u_a))$.*

We say those leftover evacuees form a *backlog*. If the height of an incoming section is less than c_j, then we use the underutilized capacity to accommodate as many of the delayed backlog evacuees as possible, together with the evacuees carried by the section. See Fig. 4, where the area of the dark-gray part equals the backlog. In this example, a section of height c_j becomes a new section in the departure section sequence out of v_{j+1}, and the light-gray part of S_3 is also a new section. Sections S_4 and S_5 maintain their shapes as they go through v_{j+1}.

At vertex $v_{j+1} = v_L(u_a)$, the first cluster in $\alpha_R^{u_b}(v_{j+1})$ would start at δ time units before all the evacuees at v_{j+1} from $P[v_L(u_a), v_R(u_a)]$ would have left, if there was no congestion on its way. Thus there would be a backlog of $w = c_j\delta$

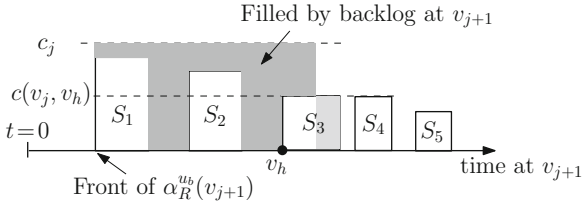


Fig. 4. $\alpha_R^{u_b}(v_{j+1})$ undergoes a change to become a part of the departure section sequence out of v_{j+1}.

evacuees still waiting at v_{j+1}. We use w arriving evacuees to fill the "space" (gaps and underutilized capacities) among initial sections in $\alpha_R^{u_b}(v_{j+1})$, as we stated before. Let $S_1, S_2, \ldots,$ be the sections of $\alpha_R^{u_b}(v_{j+1})$, which is already ceiled by c_j, and let S_h start at time t_h. Then the total amount of "space" between the first and the l^{th} sections is filled by w if

$$w \geq (t_l - t_1)c_j - \sum_{h=1}^{l-1} \lambda(S_h). \tag{7}$$

We test $l = 1, 2, \ldots$ sequentially to find up to which gap gets merged due to the backlog. The last gap (which may be after S_l and semi-infinite) is generally only partially filled.

Lemma 5. *We can construct $\mathcal{T}$ (with $\alpha_R^u(v_L(u))$ and $\beta_R^u(v_L(u))$ at every node u) in $O(n \log^2 n)$ time.*

Proof. For a node u of $\mathcal{T}$ with two child nodes u_a and u_b, we discussed above how to compute $\alpha_R^u(v_L(u))$ and $\beta_R^u(v_L(u))$, given $\alpha_R^{u_a}(v_L(u_a))$, $\beta_R^{u_a}(v_L(u_a))$, $\alpha_R^{u_b}(v_L(u_b))$, and $\beta_R^{u_b}(v_L(u_b))$. The ceiling operation of $\beta_R^{u_b}(v_L(u_b))$ by c_j, using max-heap $\mathcal{H}_u$, takes $O(|\mathcal{T}(u)| \log |\mathcal{T}(u)|)$ time, since each insertion into $\mathcal{H}_u$ takes $O(\log |\mathcal{T}(u)|)$ time, where $|\mathcal{T}(u)|$ denotes the number vertices spanned by u. The sequential tests, using (7), to find the extent of gap filling takes $O(|\mathcal{T}(u)|)$ time. Thus the total time for all nodes is $O(n \log^2 n)$. □

From now on we assume that $\mathcal{T}$ is constructed during preprocessing and available. We will make use of it in proving Lemma 6.

5 Putting Pieces Together

5.1 Computing $R(j, i)$

To run our DP, we need to compute cost $R(j, i) = \sum_h (E_h + I_h)$, where E_h (resp. I_h) is the extra cost (resp. intra cost), defined by (2), of arrival section S_h at v_{j+1}, which carries vertices of a subpath of $P[v_{j+1}, v_i]$.

Lemma 6. *If $\mathcal{T}$ is given, then for an arbitrary pair (j,i), $j<i$, we can compute $R(j,i)$ in $O((i-j)\log n)$ time.*

Proof. Let $\mathcal{P}[v_{j+1},v_i]$ denote the set of maximal subpaths of $P[v_{j+1},v_i]$ spanned by $t=O(\log n)$ nodes, $u_1,u_2,\ldots,u_t$, of $\mathcal{T}$, in this order from left to right. To compute $R(j,i)$, we combine the arrival and/or departure section sequences stored at $u_1,u_2,\ldots,u_t$ into a single arrival sequence at v_j. We discussed in Sect. 4 in detail how to combine two such sequences. Starting with $\sigma=\alpha_R^{u_1}(v_L(u_1))$, we update σ by merging it with shifted $\beta_R^{u_2}(v_L(u_2)),\beta_R^{u_3}(v_L(u_3)),\ldots$, until all of them are merged into one arrival section sequence at v_j. The shift amount for $\beta_R^{u_s}(v_L(u_s))$ is $d(v_L(u_1),v_L(u_s))\tau$. When we merge σ with $\beta_R^{u_s}(v_L(u_s))$, the capacity $c(v_j,v_L(u_s))$ must be used to ceil the shifted $\beta_R^{u_s}(v_L(u_s))$. Finding this capacity from the capacity tree $\mathcal{C}$ takes $O(\log n)$ time. The most time consuming part is testing (7) for successive l, every time two section sequences are merged. Since we must perform $O(i-j)$ such tests, the total time for all the merges is $O((i-j)\log n)$. Once the arrival section sequence at v_j is known, we can compute the intra and extra cost based on (2). □

Lemma 7. *Assuming that $\mathcal{T}$ is available, we have $t_X(n)=O(n\log^2 n)$.*

Proof. Evaluating $R(\cdot,\cdot)$ takes $O((i-j)\log n)$ time by Lemma 6, and the total time for finding switching point $x(j,j')$ is $O(n\log^2 n)$, since we need to perform binary search. □

Lemma 8. *If the edge capacities are uniform, we have $t_X(n)=O(\log^3 n)$.*

Proof. We precompute at each node u of $\mathcal{T}$ the sum of squared weights for the sections carrying the vertices spanned by u. In processing the backlog to fill gaps between the sections, the contributions from the swallowed up sections are subtracted from, and the squared weight of the new combined section is added to the sum of squared weights. Thus updating the sum of squared weights takes constant time per merging two adjacent subtrees spanning subpaths in $\mathcal{P}[v_{j+1},v_i]$. In the general capacity case, we tested (7) for successive l sequentially. But in the uniform capacity case, we can maintain the prefix sum of the gaps between successive sections. Then we can do binary search among them with backlog w to find up to which gaps are filled by w. This takes $O(\log n)$ time per merger of section sequences stored at two adjacent nodes of $\mathcal{T}$, and $O(\log^2 n)$ time for all such mergers. Thus we can find $R(j,i)$ in $O(\log^2 n)$ time, hence $t_X(n)=O(\log^3 n)$. □

5.2 Main Theorem

The correctness of our DP method can be proved similarly to [7] and the discussions above. Time complexities were analyzed in Lemmas 3, 5, 7, and 8.

Theorem 1. *(a) The minsum k-sink problem in dynamic flow path networks can be solved in $O(kn^2\log^2 n)$ time.*
(b) If the edge capacities are uniform, then it can be solved in $O(kn\log^3 n)$ time.

6 Conclusion and Discussion

We proposed an $O(kn^2 \log^2 n)$ time algorithm, based on DP that finds a minsum k-sink in dynamic flow path networks with general edge capacities, which is the first polynomial time algorithm for this problem. When the edge capacities are uniform, we also presented an $O(kn \log^3 n)$ time algorithm. There is a factor of n difference between the above two cases. The main reason is that in the general capacity case, we cannot update the intra cost in less than linear time, when merging two section sequences. We are currently working to find a way around it. A challenging problem is to efficiently solve the minsum k-sink problem in dynamic flow networks that are more general than path networks.

References

1. Arumugam, G.P., Augustine, J., Golin, M., Srikanthan, P.: A polynomial time algorithm for minimax-regret evacuation on a dynamic path. arXiv:1404,5448v1 [cs.DS], 22 April 2014, 165 (2014)
2. Bhattacharya, B., Golin, M.J., Higashikawa, Y., Kameda, T., Katoh, N.: Improved algorithms for computing k-sink on dynamic flow path networks. In: Ellen, F., Kolokolova, A., Sack, J.R. (eds.) Algorithms and Data Structures. LNCS, vol. 10389, pp. 133–144. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62127-2_12
3. Bhattacharya, B., Kameda, T.: Improved algorithms for computing minmax regret sinks on path and tree networks. Theoret. Comput. Sci. **607**, 411–425 (2015)
4. Chen, D., Golin, M.: Sink evacuation on trees with dynamic confluent flows. In: Hong, S.-H. (ed.) 27th International Symposium on Algorithms and Computation (ISAAC), Leibniz International Proceedings in Informatics, pp. 25:1–25:13 (2016)
5. Cheng, S.-W., Higashikawa, Y., Katoh, N., Ni, G., Su, B., Xu, Y.: Minimax regret 1-sink location problems in dynamic path networks. In: Chan, T.-H.H., Lau, L.C., Trevisan, L. (eds.) Theory and Applications of Models of Computation. LNCS, vol. 7876, pp. 121–132. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38236-9_12
6. Hamacher, H., Tjandra, S.: Mathematical modelling of evacuation problems: a state of the art. In: Pedestrian and Evacuation Dynamics, pp. 227–266. Springer (2002)
7. Hassin, R., Tamir, A.: Improved complexity bounds for location problems on the real line. Oper. Res. Lett. **10**(7), 395–402 (1991)
8. Higashikawa, Y., Augustine, J., Cheng, S.W., Golin, M.J., Katoh, N., Ni, G., Su, B., Xu, Y.: Minimax regret 1-sink location problem in dynamic path networks. Theoret. Comput. Sci. **588**(11), 24–36 (2015)
9. Higashikawa, Y., Golin, M.J., Katoh, N.: Minimax regret sink location problem in dynamic tree networks with uniform capacity. J. Graph Algorithms Appl. **18**(4), 539–555 (2014)
10. Higashikawa, Y., Golin, M.J., Katoh, N.: Multiple sink location problems in dynamic path networks. Theoret. Comput. Sci. **607**(1), 2–15 (2015)
11. Kariv, O., Hakimi, S.: An algorithmic approach to network location problems, Part II: the p-median. SIAM J. Appl. Math. **37**, 539–560 (1979)
12. Mamada, S., Uno, T., Makino, K., Fujishige, S.: An $O(n \log^2 n)$ algorithm for a sink location problem in dynamic tree networks. Discrete Appl. Math. **154**, 2387–2401 (2006)

Fully Leafed Induced Subtrees

Alexandre Blondin Massé[1], Julien de Carufel[2], Alain Goupil[2], Mélodie Lapointe[1], Émile Nadeau[1], and Élise Vandomme[1(✉)]

[1] Laboratoire de Combinatoire et d'Informatique Mathématique, Université du Québec à Montréal, Montreal, Canada
blondin_masse.alexandre@uqam.ca, e.vandomme@alumni.uliege.be
[2] Laboratoire Interdisciplinaire de Recherche en Imagerie et en Combinatoire, Université du Québec à Trois-Rivières, Trois-Rivières, Canada

Abstract. We consider the problem LIS of deciding whether there exists an induced subtree with exactly $i \leq n$ vertices and ℓ leaves in a given graph G with n vertices. We study the associated optimization problem, that consists in computing the maximal number of leaves, denoted by $L_G(i)$, realized by an induced subtree with i vertices, for $0 \leq i \leq n$. We begin by proving that the LIS problem is NP-complete in general. Then, we describe a nontrivial branch and bound algorithm that computes the function L_G for any simple graph G. In the special case where G is a tree of maximum degree Δ, we provide a $\mathcal{O}(n^3\Delta)$ time and $\mathcal{O}(n^2)$ space algorithm to compute the function L_G.

1 Introduction

In the past decades, subtrees of graphs, as well as their number of leaves, have been the subject of investigation from various communities. For instance in 1984, Payan *et al.* [14] discussed the maximum number of leaves, called the *leaf number*, that can be realized by a spanning tree of a given graph. This problem, called the *maximum leaf spanning tree problem* (MLST), is known to be NP-complete even in the case of regular graphs of degree 4 [13] and has attracted interest in the telecommunication network community [5,6]. The *frequent subtree mining problem* [7] investigated in the data mining community, has applications in biology. The detection of subgraph patterns such as induced subtrees is useful in information retrieval [17] and requires efficient algorithms for the enumeration of induced subtrees. In this perspective, Wasa *et al.* [16] proposed an efficient parametrized algorithm for the generation of induced subtrees in a graph.

The objects of interest in this paper are induced subtrees. The *induced* property requirement brings an interesting constraint that makes the problem significantly different from the MLST problem. A first result by Erdős *et al.* showed that the problem of finding an induced subtree of a given graph G with more

A. Blondin Massé is supported by a grant from the National Sciences and Engineering Research Council of Canada (NSERC) through Individual Discovery Grant RGPIN-417269-2013. M. Lapointe and É. Nadeau are both supported by a scholarship from the NSERC.

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 90–101, 2018.
https://doi.org/10.1007/978-3-319-94667-2_8

than i vertices is NP-complete [12]. Among induced subtrees of simple graphs, we focus on those with a maximal number of leaves. We call these objects *fully leafed induced subtrees*. Particular instances of these subtrees have recently appeared in the paper of Blondin Massé *et al.* [2], where the authors considered the maximal number of leaves that can be realized by tree-like polyominoes, respectively polycubes. The observation that tree-like polyominoes and polycubes are induced subgraphs of the lattices $\mathbb{Z}^2$ and $\mathbb{Z}^3$ respectively leads naturally to the investigation of fully leafed induced subtrees in general simple graphs, either finite or infinite.

To begin with, we consider the decision problem, called *leafed induced subtree problem (*LIS*)*, and its associated optimization problem, *maximum leafed induced subtree problem (*MLIS*)*:

*Problem 1 (*LIS*). Given a simple graph G and two positive integers i and ℓ, does there exist an induced subtree of G with i vertices and ℓ leaves?*

*Problem 2 (*MLIS*). Given a simple graph G on n vertices, what is the maximum number of leaves, $L_G(i)$, that can be realized by an induced subtree of G with i vertices, for $i \in \{0, 1, \ldots, n\}$?*

Examples of fully leafed induced subtrees are given in Fig. 1. We believe that fully leafed induced subtrees are interesting candidates for the representation of structures appearing in nature and in particular in molecular networks. Indeed, in chemical graph theory, subtrees are known to be useful in the computation of a characteristic of chemical graph, called the *Wiener index* [15]. The results of [2,15] suggest that a thorough investigation of subtrees, and in particular induced subtrees with many leaves, could lead to the discovery of combinatorial structures relevant to chemical graph theory.

This paper establishes fundamental results on fully leafed induced subtrees for further theoretical investigations and their applications. First, we prove in Sect. 2 that the problem LIS is NP-complete. To tackle the problem MLIS, we provide a branch and bound algorithm in Sect. 3. When we restrict our attention to the case of trees, it turns out that the problem MLIS is polynomial. To achieve this polynomial complexity, our proposed algorithm, described in Sect. 4, uses a dynamic programming strategy. Notice that a naive greedy approach cannot work, even in the case of trees, because a fully leafed induced subtree with n vertices is not necessarily a subtree of a fully leafed induced subtree with $n + 1$ vertices. All algorithms discussed in this paper are available, with examples, in a public GitHub repository [3]. Finally, we give in Sect. 5 some perspectives on future work. All proofs, omitted due to a lack of space, are available in the arXiv full version [1].

2 Fully Leafed Induced Subtrees

We briefly recall some definitions from graph theory. The reader is referred to [8] for graph theoretical concepts. All graphs considered in this text are simple and

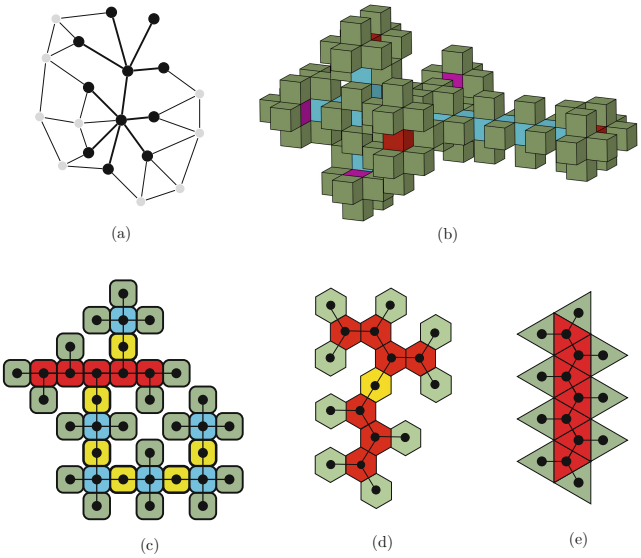


Fig. 1. Fully leafed induced subtrees in various graphs. (a) In a finite graph (the subtree of $i = 11$ vertices appears in bold). (b) In the cubic lattice. (c) In the square lattice. (d) In the hexagonal lattice. (e) In the triangular lattice. The color of each cell indicates its degree: pink for degree 5, blue for degree 4, red for degree 3, yellow for degree 2 and green for degree 1 (the leaves). (Color figure online)

undirected unless stated otherwise. Let $G = (V, E)$ be a graph with vertex set V and edge set E. The *degree* of a vertex u is the number of vertices adjacent to u and is denoted by $\deg(u)$. We denote by $|G|$ the total number $|V|$ of vertices of G and we call it the *size* of G. For $U \subseteq V$, the *subgraph of G induced by U*, denoted by $G[U]$, is the graph $G[U] = (U, E \cap \mathcal{P}_2(U))$, where $\mathcal{P}_2(U)$ is the set of all subsets of size 2 of U. Let $T = (V, E)$ be a *tree*, that is to say, a connected and acyclic graph. A vertex $u \in V$ is called a *leaf* of T when $\deg(u) = 1$. The number of leaves of T is denoted by $|T|_{\text{🍃}}$. A *subtree of G induced by U* is an induced subgraph that is also a tree.

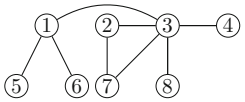
The next definitions and notation are useful in the study of the LIS and MLIS problems.

Definition 1 (Leaf function). *Given a finite or infinite graph $G = (V, E)$, let $\mathcal{T}_G(i)$ be the family of all induced subtrees of G with exactly i vertices. The* leaf function *of G, denoted by L_G, is the function with domain $\{0, 1, 2, \ldots, |G|\}$ defined by*

$$L_G(i) = \max\{|T|_{\text{🍃}} : T \in \mathcal{T}_G(i)\}.$$

As is customary, we set $\max \emptyset = -\infty$. An induced subtree T of G with i vertices is called fully leafed *when $|T|_{\text{🍃}} = L_G(i)$.*

Example 1. Consider the graph G depicted in Fig. 2 with its leaf function. The subtree induced by $U = \{1, 2, 3, 4, 6, 8\}$ is fully leafed because it has 6 vertices, 4 of them are leaves, and because $L_G(6) = 4$.



i	0	1	2	3	4	5	6	7	8
$L_G(i)$	0	0	2	2	3	4	4	5	$-\infty$

Fig. 2. A graph with vertex set $V = \{1, \ldots, 8\}$ and its leaf function.

Remark 1. For any nontrivial graph G, we have $L_G(0) = 0$ because the empty tree has no leaf, and $L_G(1) = 0$ since a single vertex is not a leaf. Finally, we always have $L_G(2) = 2$ in any graph G with at least one edge.

The following observations are immediate.

Proposition 1. *Let G be a connected graph with $n \geq 3$ vertices.*

- *If G is non-isomorphic to the complete graph on n vertices, then $L_G(3) = 2$.*
- *The sequence $(L_G(i))_{i=0,1,\ldots,|G|}$ is non-decreasing if and only if G is a tree.*

We now describe the complexity of solving the problem LIS.

Theorem 1. *The problem* LIS *of determining whether there exists an induced subtree with i vertices and ℓ leaves in a given graph is NP-complete.*

To show that it is NP-complete we reduce it to the well-know NP-complete independent set problem (IndependentSet) [13]: *Given a graph G and a positive integer k, does there exist an independent set of size k in G, i.e. a subset of k vertices that are not pairwise adjacent?* For the reduction, we consider the map that associates to an instance (G, k) of IndependentSet, the instance $(H, k + 1, k)$ of LIS such that the graph H is a copy of G with an additional universal vertex u that is linked to each vertex of G, $k + 1$ is the number of vertices of the induced subtree of H and k is its number of leaves.

From this reduction, we obtain insights on the parameterized complexity of LIS problem. A problem, which is parameterized by $k_1, \ldots, k_j$, is said to be *fixed parameter tractable* if it can be solved in time $\mathcal{O}(f(k_1, \ldots, k_j)n^c)$ where n is the size of the input, c is a constant independent from the parameters $k_1, \ldots, k_j$ and f is a function of $k_1, \ldots, k_j$. The class FPT contains all parameterized problems that are *fixed parameter tractable*. Similarly to the conventional complexity theory, Downey and Fellows introduced a hierarchy of complexity classes to describe the complexity of parameterized problems [11]: FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \ldots$ Since IndependentSet is W[1]-complete [9], it follows that LIS is fixed parameter intractable, unless FPT = W[1]. Note that when we replace the "induced" condition with "spanning", the problem becomes fixed parameter tractable [4,10].

Corollary 1. *If* FTP $\neq$ W[1]*, then* LIS $\notin$ FPT*.*

We end this section with results on leaf functions for particular families of graphs. We can easily compute the leaf function for cycle, wheel and complete graphs as well as for complete bipartite graphs [1]. For the hypercube graph Q_d with 2^d vertices, the computation of L_{Q_d} is more intricate. Using the branch and bound algorithm described in Sect. 3 and implemented in [3], we were able to compute the values of the function L_{Q_d} for $d \leq 6$ (see [1]).

Some infinite lattices were studied by Blondin Massé *et al.* [2]. They have computed the leaf functions of the regular square lattice $\mathbb{Z}^2$ and of the regular hexagonal and triangular lattices. They also studied the regular cubic lattice $\mathbb{Z}^3$. These leaf functions always satisfies a linear recurrence, with asymptotic growth $i/2$ for planar lattices and with asymptotic growth $28i/41$ for the cubic lattice.

3 Computing the Leaf Function of a Graph

We now describe a branch and bound algorithm that computes the leaf function $L_G(i)$ for an arbitrary graph G. We propose an algorithm based on a data structure that we call an *induced subtree configuration*.

Definition 2. *Let* $G = (V, E)$ *be a graph and* $\Gamma = \{\text{green}, \text{yellow}, \text{red}, \text{blue}\}$ *be a set of colors. An* induced subtree configuration *of* G *is an ordered pair* $C = (c, H)$*, where* $c : V \to \Gamma$ *is a coloring function and* H *is a stack of colorings called the* history *of* C*.*

All colorings $c : V \to \Gamma$ *must satisfy the following conditions for any* $u, v \in V$*:*

(i) The subgraph induced by $c^{-1}(\text{green})$ *is a tree;*
(ii) If $c(u) = \text{green}$ *and* $\{u, v\} \in E$*, then* $c(v) \in \{\text{green}, \text{yellow}, \text{red}\}$*;*
(iii) If $c(u) = \text{yellow}$*, then* $|c^{-1}(\text{green}) \cap N(u)| = 1$*, where* $N(u)$ *denotes the set of neighbors of* u*.*

The initial induced subtree configuration *of a graph* G *is the pair* (c_{blue}, H) *where* $c_{\text{blue}}(v) = \text{blue}$ *for all* $v \in G$ *and* H *is the empty stack. When the context is clear,* C *is simply called a* configuration*. See Fig. 3(a) for an example.*

Roughly speaking, a configuration is an induced subtree enriched with information that allows one to generate other induced subtrees either by extension, by exclusion or by backtracking. The colors assigned to the vertices can be interpreted as follow. The *green* vertices are the confirmed vertices to be included in a subtree. Since each *yellow* vertex is connected to exactly one *green* vertex, any *yellow* vertex can be safely added to the green subtree to create a new induced subtree. A *red* vertex is excluded of the subtree. The exclusion of a *red* vertex is done either because it is adjacent to more than one *green* vertex and its addition would create a cycle or because it is explicitly excluded for generation purposes. Finally, the *blue* vertices are available vertices that have not yet been considered and that could be considered later. For reasons that are explained in the next

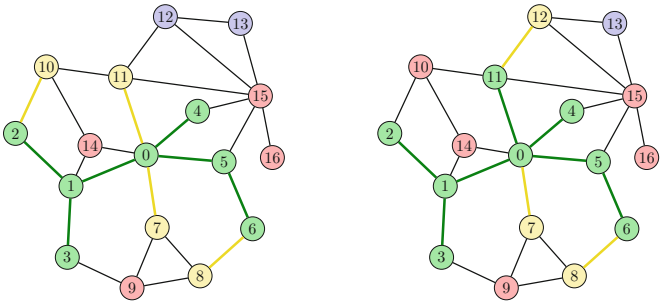


Fig. 3. Induced subtree configurations. The green edges outline the induced subtree and the yellow edges outline its possible extensions. (a) A configuration C. (b) The configuration C.AddToSubtree(11). (Color figure online)

paragraphs, it is convenient to save in the stack H the colorations from which C was obtained.

Let $C = (c, H)$ be a configuration of some graph $G = (V, E)$, with coloring c and stack H. We consider the following operations on C:

- C.VertexToAdd() is a non deterministic function that returns any non *green* vertex in G that can be safely colored in *green*. If no such vertex exists, it returns *none*. Note that the color of the returned vertex is always *yellow*, except when $c^{-1}(green) = \emptyset$, where the color is *blue*.
- C.AddToSubtree(v) first pushes a copy of c on top of H, sets the color of v to *green* and updates the colors of the neighborhood of v accordingly. Notice that this operation is applied only to a vertex v that can be safely colored in green.
- C.ExcludeVertex(v) first pushes a copy of c on top of H and then sets the color of v to *red*. This operation is applied only on a vertex v such that $c(v) \in \{yellow, blue\}$.
- C.Undo() retrieves and removes the top of H, then stores it into c. In other words, this operation cancels the last operation applied on C, which is either an inclusion or an exclusion.

It is quite straightforward to use configurations for the generation of all induced subtrees of a graph G. Starting with the initial configuration, it is sufficient to recursively build configurations by branching according to whether some vertex v returned by the operation C.VertexToAdd() is included or excluded from the current *green* tree. Considering this process as a tree of configurations, the operation can be paired with edges of this tree. Therefore, a careful analysis shows that the generation runs in $\mathcal{O}(|V|)$ amortized per solution and $\mathcal{O}(|V|)$ in space.

While iterating over all possible configurations, it is obvious that some configurations should be discarded whenever they cannot extend to configurations

with many leaves. Therefore, given a configuration C, we define the function C.LeafPotential(n') which computes an upper bound on the number of leaves that can be reached by extending C to a configuration of n' *green* vertices. First, the potential is $-\infty$ for n' greater than the size of the connected component K containing the green subtree (when computing the connected component, we treat red vertices as removed from the graph). Second, in order to compute this upper bound for $n' \leq |K|$, we consider an optimistic scenario in which all available *yellow* and *blue* vertices that are close enough can safely be colored in *green*. Keeping this idea in mind, we start by partitioning the *available* vertices, which are the *yellow* and *blue* vertices together with the leaves of the *green* tree, according to their distance from the inner vertices of the configuration subtree in K. Algorithm 1 describes these steps in details.

Algorithm 1 Computation of the leaf potential for n'

```
1: function LeafPotential(C : configuration, n' : natural): natural
2:     n ← number of green vertices; ℓ ← number of leaves in the green subtree
3:     y ← number of yellow vertices adjacent to an inner green subtree vertex
4:     if n + y ≥ n' then
5:         (n, ℓ) ← (n', ℓ + (n' − n))
6:     else
7:         (n, ℓ) ← (n + y, ℓ + y)
8:     end if
9:     d ← 1
10:    while n < n' and there exists an available vertex at distance at most d do
11:        Let v be an available vertex of highest degree
12:                                   ▷ The degree does not count red vertices
13:        if n + deg(v) − 1 ≤ n' then
14:            (n, ℓ) ← (n + deg(v) − 1, ℓ + deg(v) − 2)
15:        else
16:            (n, ℓ) ← (n', ℓ + (n' − n) − 1)
17:        end if
18:        Remove v from available vertices; d ← d + 1
19:    end while
20:    return ℓ
21: end function
```

The first part of Algorithm 1 consists in *completing* the *green* subtree. More precisely, a configuration C is called *complete* if each *yellow* vertex is adjacent to a leaf of the *green* tree. We first verify if C is complete and, when it is not the case, we increase n and ℓ as if the *green* subtree was completed (Lines 4–8). Next, we choose a vertex v among all available vertices within distance d. We assume that v is *green* and update n and ℓ as if all non-*green* neighbors of v were leaves added to the current configuration (Lines 13–17). This process is repeated until the size of the "optimistic subtree" reaches n'.

One can prove that Algorithm 1 yields an upper bound on the maximum number of leaves that can be realized. It is worth mentioning that, in order to

obtain a nontrivial bound, we restrict the available vertices to those that are within distance d from the inner vertices of the current *green* subtree, and then we increase the value of d at each iteration.

Proposition 2. *Let C be a configuration of a graph $G = (V, E)$ with $n \geq 3$ green vertices and let n' be an integer such that $n \leq n' \leq |V|$. Then any extension of C to a configuration of n' vertices has at most $C.\textsc{LeafPotential}(n')$ leaves, where $C.\textsc{LeafPotential}(n')$ is the operator described in Algorithm 1.*

It follows from Proposition 2 that a configuration C of n *green* vertices and r *red* vertices cannot be extended to a configuration whose subtree has more leaves than prescribed by the best values found for L so far when

$$C.\textsc{LeafPotential}(n') \leq L(n') \text{ for all } n \leq n' \leq |K|. \tag{1}$$

Based on Proposition 2 and Eq. (1), we can adapt the induced subtree generator to a branch and bound algorithm that discards useless configurations.

Theorem 2. *Let G be a graph. Then the branch and bound algorithm described in the previous paragraphs of the present section computes the leaf function L_G of G.*

Empirically, we observed the following elements. First, it seems that the overall time performance is significantly better on dense graphs. More precisely, for a fixed number of vertices, the computation of the leaf function is faster on a dense graph than on a sparse one (see Fig. 4(a–b)). This is not surprising, since if one takes a vertices subset of a dense graph, the probability that these vertices induce at least one cycle is high. Therefore, the number of visited induced subtrees is smaller. For example, experimental data show that the number of visited subtrees in a graph with 30 vertices and density 0.1 is still around ten times greater than the number of visited subtrees in a graph with 80 vertices and density 0.9.

Moreover, the leaf potential bound always reduces the number of visited subtrees regardless of the density. However, the difference is more pronounced on lower density graphs (see Fig. 4(c–d)). This also seems easily explainable: It is expected that, as the density decreases, the number of layers in the vertices partition increases and the degrees of the vertices diminish. Hence, when we use the leaf potential as a bounding strategy, the computation time gain is more significant on sparse graphs.

Hence, for lower density graphs, the leaf potential improves the algorithm and the overall performance of the algorithm. For higher density, no significant difference in time performance with or without the usage of the bound is observed. Finally, from an empirical point of view, the number of visited induced subtrees seems to indicate an overall complexity of the algorithm in $\mathcal{O}(\alpha^n)$ with $\alpha < 2$. Unfortunately, we were unable to prove such an upper bound.

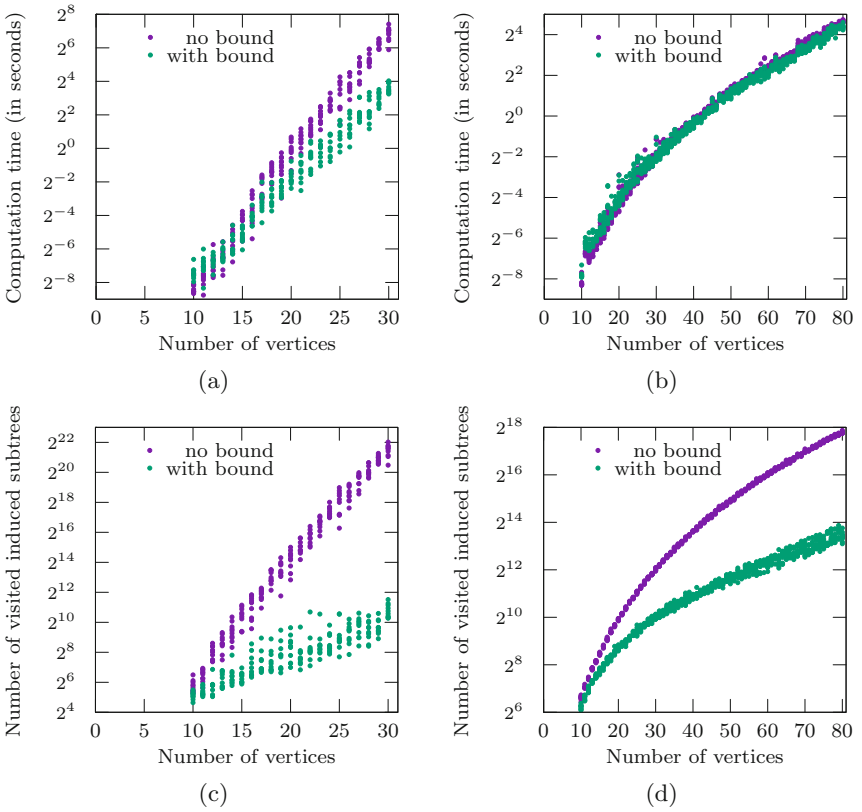


Fig. 4. The running time of the branch and bound algorithm on 10 randomly generated graphs with density 0.2 (a) and density 0.8 (b), with or without using the leaf potential bound. The corresponding number of induced subtrees for density 0.2 (c) and density 0.8 (d) that are visited during the execution. (Color figure online)

4 Fully Leafed Induced Subtrees of Trees

It turns out that the MLIS problem can be solved in polynomial time when it is restricted to the class of trees. Observe that since all subtrees of trees are induced subgraphs, we can omit the "induced" adjective.

A naive strategy consists in successively deleting suitable leaves to obtain a sequence of fully leafed subtrees embedded in each other. Such a strategy is not viable. Indeed, consider the tree T represented in Fig. 5. We have $L_T(9) = 6$ and $L_T(7) = 5$ and there is exactly one fully leafed induced subtree of T with respectively 7 and 9 vertices. But the smallest of these two subtrees (in blue) is not a subgraph of the largest one (in red).

Hereafter, we describe an algorithm with polynomial time complexity based on the dynamic programming paradigm but before, we recall some definitions. A *rooted tree* is a couple $\widehat{T} = (T, u)$ where $T = (V, E)$ is a tree and $u \in V$ is a

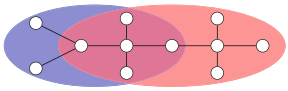


Fig. 5. A tree with its unique fully leafed induced subtrees with 7 (respectively 9) vertices in the blue (resp. red) area. (Color figure online)

distinguished vertex called the *root* of $\widehat{T}$. Rooted trees have a natural orientation with arcs pointing away from the root. A *leaf* of a rooted tree is a vertex v with outdegree $\deg^+(v) = 0$. In particular, if a rooted tree consists in a single vertex, then this vertex is a leaf. The functions $|\widehat{T}|$ and $|\widehat{T}|_{\text{leaf}}$ are defined accordingly by

$$|\widehat{T}| = |T| \text{ and } |\widehat{T}|_{\text{leaf}} = \left|\left\{v \in \widehat{T} : \deg^+(v) = 0\right\}\right|.$$

Similarly, a *rooted forest* $\widehat{F}$ is a collection of rooted trees. It follows naturally that

$$|\widehat{F}|_{\text{leaf}} = \sum_{\widehat{T} \in \widehat{F}} |\widehat{T}|_{\text{leaf}}.$$

The *rooted forest induced by a rooted tree* $\widehat{T} = (T, u)$ is the set of rooted trees obtained by removing from T the root u and its incident edges so that the k vertices adjacent to u become roots of the trees $\widehat{T}_i$. Let $\widehat{T}$ be any rooted tree with n vertices and $L_{\widehat{T}} : \{0, 1, \ldots, n\} \to \mathbb{N}$ be defined by

$$L_{\widehat{T}}(i) = \max\left\{|\widehat{T'}|_{\text{leaf}} \mid \widehat{T'} \preceq \widehat{T} \text{ and } |\widehat{T'}| = i\right\},$$

where $\preceq$ denotes the relation "being a rooted subtree with the same root". Roughly speaking, $L_{\widehat{T}}(i)$ is the maximum number of leaves that can be realized by some rooted subtree of size i of $\widehat{T}$. This map is naturally extended to rooted forests. Thus for a rooted forest $\widehat{F} = \{\widehat{T_1}, \ldots, \widehat{T_k}\}$ we set

$$L_{\widehat{F}}(i) = \max\left\{\sum_{j=1}^{k} |\widehat{T'_j}|_{\text{leaf}} \mid \widehat{T'_j} \preceq \widehat{T_j} \text{ and } \sum_{j=1}^{k} |\widehat{T'_j}| = i\right\}. \tag{2}$$

Let $C(i, k)$ be the set of all weak compositions $\lambda = (\lambda_1, \ldots, \lambda_k)$ of i in k non-negative parts. Then Eq. (2) is equivalent to

$$L_{\widehat{F}}(i) = \max\left\{\sum_{j=1}^{k} L_{\widehat{T_j}}(\lambda_j) \;:\; \lambda \in C(i, k)\right\}. \tag{3}$$

Assuming that $L_{\widehat{T_j}}$ is known for $j = 1, 2, \ldots, k$, a naive computation of $L_{\widehat{F}}$ using Eq. (3) is not done in polynomial time, since $|C(i, k)| = \binom{i+k-1}{i}$. Nevertheless, the next lemma shows that $L_{\widehat{F}}$ can be computed in polynomial time.

Lemma 1. *Let $k \geq 1$ be an integer and $\widehat{F} = \{\widehat{T_1}, \ldots, \widehat{T_k}\}$ be a rooted forest with n vertices. Then, for $i \in \{0, \ldots, n\}$, we have $L_{\widehat{F}}(i) = L_{\widehat{T_1}}(i)$ if $k = 1$, and if $k \geq 2$*

$$L_{\widehat{F}}(i) = \max\{L_{\widehat{T_1}}(j) + L_{\widehat{F'}}(i-j) : \max\{0, i - |\widehat{F'}|\} \leq j \leq \min\{i, |\widehat{T_1}|\}\}$$

where $\widehat{F'} = \{\widehat{T_2}, \ldots, \widehat{T_k}\}$. Therefore, if $L_{\widehat{T_j}}$ is known for $j = 1, 2, \ldots, k$, then $L_{\widehat{F}}$ can be computed in $\mathcal{O}(kn^2)$ time.

Finally, we describe how $L_{\widehat{T}}$ is computed from the children of its root.

Lemma 2. *Let $\widehat{T}$ be some rooted tree with root u. Let $\widehat{F}$ be the rooted forest induced by the children of u. Then $L_{\widehat{T}}(i) = i$ if $0 \leq i \leq 1$ and,*

$$L_{\widehat{T}}(i) = L_{\widehat{F}}(i-1), \quad \text{if } 2 \leq i \leq |\widehat{T}|.$$

Combining Lemmas 1 and 2, we obtain the following result.

Theorem 3. *Let $T = (V, E)$ be an unrooted tree with $n \geq 2$ vertices. Then L_T can be computed in $\mathcal{O}(n^3\Delta)$ time and $\mathcal{O}(n^2)$ space where Δ denotes the maximal degree of a vertex in T.*

Remark 2. At first sight, one might think that a more careful analysis could lead to a $\mathcal{O}(n^3)$ time complexity in Theorem 3. However, we have not been able to get rid of the Δ factor for the following reason. Consider a tree $\widehat{T}$ rooted in u and the forest $\widehat{F}$ induced by $\widehat{T}$. By Lemma 1, the computation of $L_{\widehat{F}}$ requires $\deg(u) - 1$ "merging steps", i.e., computations using the recursive part of the formula. Since, in the graph, each arc incident to u induces a different rooted tree and an associated rooted forest, it does not seem possible to reuse the merger of one forest in the computation of another. Therefore, the number of mergers of a given edge can increase up to Δ.

5 Perspectives

There is room for improving and specializing the branch and bound algorithm described in Sect. 3. For example, we were able to speed up the computations for the hypercube Q_6 by taking into account some symmetries (see [3]). In a more general context, we believe that significant improvements could be obtained by exploiting the complete automorphism group of the graph, particularly in highly symmetric graphs.

Due to a lack of space, we did not discuss the problem of *generating* efficiently the set of all fully leafed induced subtrees. However, it seems easy to show that, by slightly modifying the branch and bound algorithm and the dynamic programming approach of Sect. 4, one could generate all optimal induced subtrees with polynomial time delay.

Finally, since the problem MLIS is polynomial for trees, another possible study would be to restrict our attention to special families of graphs. The classes of 3-colorable graphs, planar graphs and chordal graphs seem promising for finding a polynomial time algorithm, as well as the family of graphs with bounded tree-width.

References

1. Blondin Massé, A., de Carufel, J., Goupil, A., Lapointe, M., Nadeau, É., Vandomme, É.: Fully leafed induced subtrees (2017). arXiv.org/abs/1709.09808
2. Blondin Massé, A., de Carufel, J., Goupil, A., Samson, M.: Fully leafed tree-like polyominoes and polycubes. In: Brankovic, L., Ryan, J., Smyth, W.F. (eds.) IWOCA 2017. LNCS, vol. 10765, pp. 206–218. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78825-8_17
3. Blondin Massé, A., Nadeau, É.: Fully leafed induced subtrees, GitHub Repository. https://github.com/enadeau/fully-leafed-induced-subtrees
4. Bodlaender, H.L.: On linear time minor tests and depth first search. In: Dehne, F., Sack, J.-R., Santoro, N. (eds.) WADS 1989. LNCS, vol. 382, pp. 577–590. Springer, Heidelberg (1989). https://doi.org/10.1007/3-540-51542-9_48
5. Boukerche, A., Cheng, X., Linus, J.: A performance evaluation of a novel energy-aware data-centric routing algorithm in wireless sensor networks. Wirel. Netw. **11**(5), 619–635 (2005)
6. Chen, S., Ljubić, I., Raghavan, S.: The generalized regenerator location problem. INFORMS J. Comput. **27**(2), 204–220 (2015)
7. Deepak, A., Fernández-Baca, D., Tirthapura, S., Sanderson, M.J., McMahon, M.M.: EvoMiner: frequent subtree mining in phylogenetic databases. Knowl. Inf. Syst. **41**(3), 559–590 (2014)
8. Diestel, R.: Graph Theory. Graduate Texts in Mathematics, vol. 173, 4th edn. Springer, Heidelberg (2010)
9. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness II: on completeness for W[1]. Theoret. Comput. Sci. **141**(1), 109–131 (1995)
10. Downey, R.G., Fellows, M.R.: Parameterized computational feasibility. In: Clote, P., Remmel, J.B. (eds.) Feasible Mathematics II, pp. 219–244. Birkhäuser Boston, Boston (1995)
11. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Monographs in Computer Science. Springer, New York (1999). https://doi.org/10.1007/978-1-4612-0515-9
12. Erdős, P., Saks, M., Sós, V.T.: Maximum induced trees in graphs. J. Combin. Theory Ser. B **41**(1), 61–79 (1986)
13. Garey, M.R., Johnson, D.S.: Computers and Intractability. W. H. Freeman and Co., San Francisco (1979)
14. Payan, C., Tchuente, M., Xuong, N.H.: Arbres avec un nombre maximum de sommets pendants (Trees with a maximal number of vertices with degree 1). Discrete Math. **49**(3), 267–273 (1984)
15. Székely, L.A., Wang, H.: On subtrees of trees. Adv. Appl. Math. **34**(1), 138–155 (2005)
16. Wasa, K., Arimura, H., Uno, T.: Efficient enumeration of induced subtrees in a K-degenerate graph. In: Ahn, H.-K., Shin, C.-S. (eds.) ISAAC 2014. LNCS, vol. 8889, pp. 94–102. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13075-0_8
17. Zaki, M.J.: Efficiently mining frequent trees in a forest. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2002, pp. 71–80. ACM, New York (2002)

Pattern Matching for k-Track Permutations

Laurent Bulteau[1], Romeo Rizzi[2], and Stéphane Vialette[1(✉)]

[1] Université Paris-Est, LIGM (UMR 8049), CNRS, ENPC, ESIEE Paris, UPEM, 77454 Marne-la-Vallée, France
{laurent.bulteau,stephane.vialette}@u-pem.fr
[2] Department of Computer Science, University of Verona, Verona, Italy
romeo.rizzi@univr.it

Abstract. Given permutations τ and π, the permutation pattern (PP) problem is to decide whether π occurs in τ as an order-isomorphic subsequence. Although an FPT algorithm is known for PP parameterized by the size of the pattern $|\pi|$ [Guillemot and Marx 2014], the high complexity of this algorithm makes it impractical for most instances. In this paper we approach the PP problem from k-track permutations, *i.e.* those permutations that are the union of k increasing patterns or, equivalently, those permutation that avoid the decreasing pattern $(k+1)k\ldots1$. Recently, k-track permutations have been shown to be central combinatorial objects in the study of the PP problem. Indeed, the PP problem is NP-complete when π is 321-avoiding and τ is 4321-avoiding but is solvable in polynomial-time if both π and τ avoid 321. We propose and implement an exact algorithm, FPT for parameters k and $|\pi|$, which allows to solve efficiently some large instances.

1 Introduction

A target permutation τ is said to contain the pattern (shorter permutation) π, in symbols $\pi \preceq \tau$, if there exists an order-preserving embedding of π into τ (*i.e.*, if there exists a subsequence of entries of τ that has the same relative order as π); alternatively, π is said to be involved in τ. Otherwise, τ is said to avoid π. For example, $\tau = 3215674$ contains the pattern $\pi = 132$ since the subsequence 154 is ordered in the same way as 132. See Fig. 1 for another example. Pattern involvement in permutations has become a very active area of research.

We consider here the Permutation Pattern (PP) problem. Given two permutations τ and π, the PP problem is to decide whether π is involved in τ (the problem is ascribed to Wilf in [6]). The PP problem is NP-hard [6], but is solvable in $O^*(n^m)$ time if π has size m and τ has size n. Improvements were presented in [2] and [1], the latter describing an $O(n^{0.47m+o(m)})$ time algorithm. Of particular importance, Marx and Guillemot [12] proved the problem to be fixed-parameter tractable (FPT) for parameter m (the running time of the algorithm is $n\ 2^{O(m^2\log(m))}$). However, this result is mainly of theoretical interest, since, to the best of our understanding, it yields an algorithm with a rather prohibitive running time. Taking advantage of the structure of the permutations, an

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 102–114, 2018.
https://doi.org/10.1007/978-3-319-94667-2_9

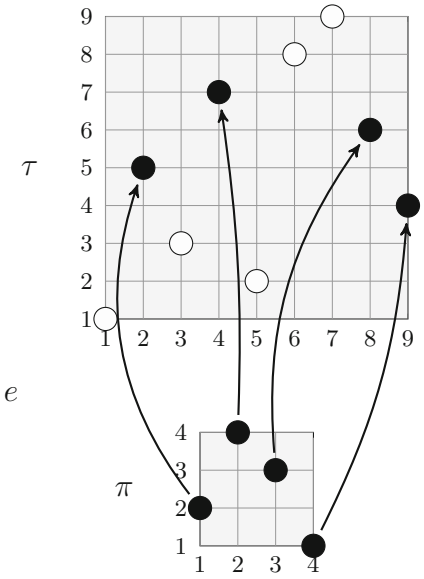


Fig. 1. An embedding of $\pi = 2431$ into $\tau = 153728964$.

$O(1.79^{\mathrm{run}(\tau)}\, nm)$ time algorithm has been presented in [9], where $\mathrm{run}(\tau)$ is the number of alternating runs in τ. The PP problem is known to be polynomial-time solvable (in m and n) if pi is separable (*i.e.* π contains neither the pattern 2413 nor 3142) [6,14,18]. In case π is monotone, an $O(n \log\log m)$ time algorithm is known [10]. Finally, notice that in the last years, other types of patterns in permutations have been studied: vincular, bivincular and mesh patterns, just to name a few (see [8,16] and references therein).

In this paper, we focus on k-track permutations, *i.e.*, those permutations that can be decomposed into k increasing subsequences. For example 14576238 is a 3-track permutation as it can be decomposed into 1578, 46 and 23. Such permutations are commonly known as $(k+1)k \ldots 1$-avoiding permutations. Pattern matching for 321-avoiding permutations have been considered in [13] and [3], the latter presenting a fast polynomial-time algorithm in case both π and τ are 321-avoiding. It has been recently proven that the PP problem remains NP-complete when π is 321-avoiding and τ is 4321-avoiding [17]. Strongly related in this setting is the NP-complete problem of partitioning a permutation into a minimum number of monotone subsequences [4,7,15,23].

Erdős and Szekeres theorem of 1935 states that every permutation of n distinct reals contains a monotone subsequence of length $\lceil \sqrt{n} \rceil$, see the review [21]. Greedily extracting longest monotone subsequences in an iterative way yields a partition into at most $2\lfloor \sqrt{n} \rfloor$ monotone subsequences (an $O\left(n^{1.5}\right)$ time procedure). Note, however, finding a minimum size partition into monotone subsequences is NP-hard [24]. For fixed k and ℓ (not part of the input), a parti-

tion into exactly k increasing and ℓ decreasing subsequences can be computed in $O\left(n^{k+\ell}\right)$ time [7]. A minimum monotone partition can be approximated within a factor of 1.71 in $O\left(n^{2.5}\right)$ time, see [11]. The natural generalization asking for partitions into k-modal subsequences (*i.e.*, sequences having at most k internal local extrema) is considered in [22]. As $n \to \infty$, for inputs in which each permutation of the input is equally likely, the expected length of the longest increasing subsequence is $2\sqrt{n} + cn^{1/6} + o(n^{1/6})$, where $c = -1.77108\ldots$ is a constant having a complicated definition in terms of the solution to a certain differential equation, the Painlevé equation of type II [5] (see also [20]).

We show in this paper that searching for a pattern into a k-track permutation can be significantly sped-up when k is small, by providing a FPTalgorithm with parameters m and k. More precisely, our algorithm runs in time $O(nk^m m^3)$, and experiments allowed us to solve efficiently enough instances with $n = 600$, $m = 20$ and $k = 6$. An informal description of our algorithm is as follows. We first fix a partition (*i.e.*, a coloring) of the target permutation τ and guess, for each element in π, the track on which it should be searched. We map the elements of π onto the first positions of τ of the corresponding track. Whenever incompatibilities appear in the mapping (either a higher-index element of π is mapped to a lower-index element of τ, or a higher-value element of π is mapped to a lower-value element of τ), an element is "*pushed*" forward on its own track, until no more conflicts exist (then a pattern is found), or the end of a track is reached (and we move on to the next guessing, if any). The main bottleneck of this algorithm is clearly the exponential enumeration of the colorings of the pattern. We thus use a specific coloring of the target permutation in order to reduce the search space.

This paper is organized as follows. Section 2 progressively introduces the needed definitions and properties. Section 3 is devoted to presenting the proposed algorithm, and benchmarks are presented in Sect. 4.

2 Step-by-Step: Definitions and Propositions

2.1 Coloring

We first define the *coloring* and number of tracks of a permutations. Informally, a k-coloring is a partition of a permutation into k increasing subsequences, see Fig. 2(a).

Definition 1 (k-coloring). *A* k-coloring *of a permutation* π *of* $[n]$ *is a function* $c : [n] \to [k]$*, such that, for each* $i, j \in [n]$*,* $i < j$*, we have* $c(i) = c(j) \Rightarrow \pi[i] < \pi[j]$*.*

Definition 2 (k-track). *A permutation is a* k-track *permutation if it admits a* k*-coloring.*

A folklore property is that π is a k-track permutation if and only if it does not contain a decreasing subsequence of length $k + 1$.

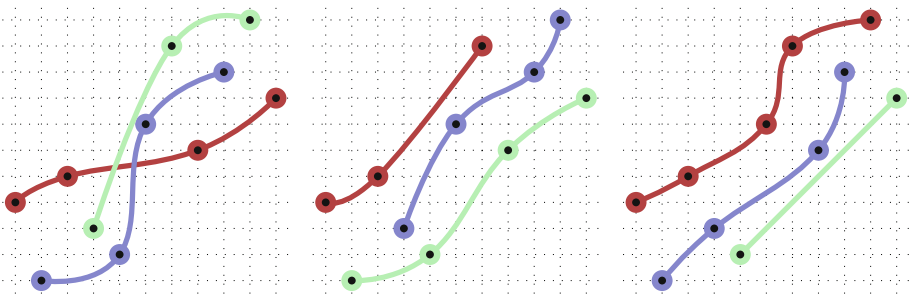


Fig. 2. Three possible colorings of the same permutation $415327A69B8$ positions with the same color are linked by a path: (a) an arbitrary coloring (the only constraint is that the paths are increasing), (b) an ordered coloring (where any decreasing pair of positions have increasing colors, i.e. paths are non-crossings) and (c) the canonical coloring (where the color is determined by the decreasing subsequences). (Color figure online)

Definition 3 (Colored permutation). *A* colored permutation *is a pair* (π, c) *where* c *is a coloring of* π.

Note that a k-track permutation is also a k'-track permutation for all $k' > k$.

2.2 Embedding

Definition 4 (Pre-embedding and embedding). *Given two permutations,* π *of* $[m]$ *and* τ *of* $[n]$*, a* pre-embedding *of* π *into* τ *is any function* $e : [m] \to [n]$*. A pair* (i, j) *is an* order *(resp.* value*)* conflict *for* e *if* $i < j$ *and* $e(i) \geq e(j)$ *(resp.* $\pi[i] < \pi[j]$ *and* $\tau[e(i)] \geq \tau[e(j)]$*). An* embedding *is a pre-embedding without any order nor value conflict.*

If we restrict ourselves to color-preserving embeddings (for some fixed coloring of π and τ), then the problem of finding an embedding actually becomes simpler. We focus on this special case.

Definition 5 (Color preserving (pre-)embedding). *Given two colored permutations,* (π, c) *and* (τ, c')*, a* color preserving (pre-)embedding *of* (π, c) *into* (τ, c') *is a (pre-)embedding* e *of* π *into* τ *such that* $c'(e(i)) = c(i)$ *for all* $i \in [m]$.

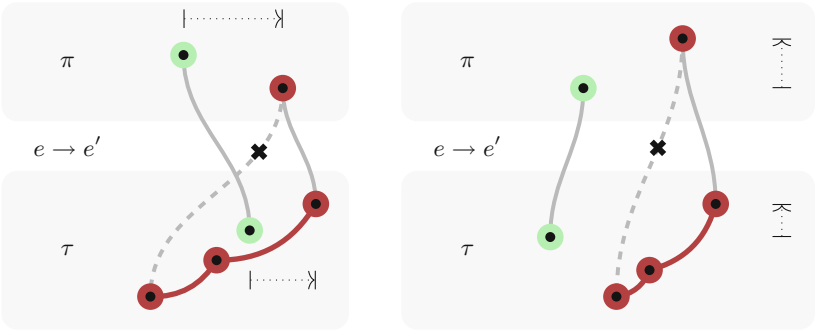
If there exits an embedding e of permutations π into τ, and τ admits a k-coloring c, then it is straightforward to verify that $c \circ e$ is a k-coloring of τ, it is called the *inherited coloring* of π for e from coloring c.

2.3 Leftmost Pre-embeddings

Leftmost color preserving pre-embeddings are at the heart of our approach. Given a fixed k-coloring of τ, we build a color-preserving embedding by starting

with a pre-embedding on which we progressively solve conflicts. We maintain the invariant that the current pre-embedding is always a lower-bound, in the following sense, of any possible embedding (for the same coloring).

Definition 6 (Leftmost color preserving pre-embedding). *Given two colored permutations, a color preserving pre-embedding e is* leftmost *if any colored embedding e^* is such that $e^*(i) \geq e(i)$ for all $i \in [m]$.*



Rule 1 Rule 2

Fig. 3. Rules 1 and 2. If two elements of π create an order (resp. value) conflict, match the rightmost (resp. top-most) element to the first possible position in τ, of the same color, that solves the conflict.

We now give two rules (see Fig. 3) applying to any leftmost colored pre-embedding e of (π, c_π) into (τ, c_τ). We prove for each one that it is sound: (i) it returns "*no*" only if there is no embedding of (π, c_π) into (τ, c_τ), and (ii) if it returns e', then e' is a leftmost pre-embedding such that $e' \neq e$ and $e'(i) \geq e(i)$ for all $i \in [m]$.

Rule 1 (Solving order conflicts) *If e has an order conflict (i_0, j_0), then build e' such that $e'(i) = e(i)$ for all $i \neq j_0$, and $e'(j_0) = \min\{u > e(i_0) : c_\tau(u) = c_\pi(j_0)\}$, or return "*no*" if this set is empty*

Proposition 1. *Rule 1 is sound.*

Proof. Assume that there exists a colored embedding e^* of (π, c_π) into (τ, c_τ). Since e is Leftmost, we have $e^*(i) \geq e(i)$ for all i. Hence $e^*(i) \geq e'(i)$ for all $i \neq j_0$. For $i = j_0$, since e^* is increasing, $e^*(j_0) > e^*(i_0) \geq e(i_0)$. Since $c_\tau(e^*(j_0)) = c_\pi(j_0)$ (since e^* is colored), then $e^*(j_0) \in \{u > e(i_0) \mid c_\tau(u) = c_\pi(j_0)\}$. Then this set is non-empty (the rule does not return "*no*") and $e'(j_0) \leq e^*(j_0)$.

Rule 2 (Solving value conflicts) *If e has a value conflict (i_0, j_0), then build e' such that $e'(i) = e(i)$ for all $i \neq j_0$, and $e'(j_0) = \min\{u > e(j_0) \mid \tau[u] > \tau[e(i_0)]$ and $c_\tau(u) = c_\pi(j_0)\}$, or return "*no*" if this set is empty.*

Proposition 2. *Rule 2 is sound.*

Proof. Assume that there exists a colored embedding e^* of (π, c_π) into (τ, c_τ). Since e is Leftmost, we have $e^*(i) \geq e(i)$ for all i. Hence $e^*(i) \geq e'(i)$ for all $i \neq j_0$. For $i = j_0$, since (i_0, j_0) is not a value conflict for e^*, then $\tau[e^*(j_0)] > \tau[e^*(i_0)] \geq \tau[e(i_0)] \geq \tau[e(j_0)]$. Note in particular that $e^*(j_0) \neq e(j_0)$ (since $\tau[e^*(j_0)] \neq \tau[e(j_0)]$). Moreover, we have $c_\tau(e^*(j_0)) = c_\pi(j_0)$ (since e^* is colored), so overall $e^*(j_0) \in \{u > e(j_0) \mid \tau[u] > \tau[e(i_0)] \text{ and } c_\tau(u) = c_\pi(j_0)\}$. Then this set is non-empty (the rule does not return "*no*") and $e'(j_0) \leq e^*(j_0)$.

Notice that resolving conflicts (*i.e.*, applying Rules 1 and 2) may introduce new conflicts. Since a conflict-free pre-embedding is an embedding, we have the following result.

Proposition 3. *If a colored pre-embedding e does not satisfy the conditions of Rules 1 and 2, then e is a colored embedding.*

Proof. This is trivial since, by definition, a conflict-free pre-embedding is an embedding.

2.4 Canonical Coloring

A k-track permutation may have many different k-colorings. For our purpose, we make use of the following specific coloring, see Fig. 2(c).

Definition 7 (Canonical coloring). *Given a permutation π, the* canonical coloring *of π, denoted can_π, is the coloring such that, for all $i \in [n]$, $\mathrm{can}_\pi(i)$ is the length of the longest decreasing subsequence of π ending at position i.*

Note that the above definition is a coloring. Indeed, if $i < j$ and $\pi[i] \geq \pi[j]$, then any decreasing sequence ending in position i can be extended once with position j, so $\mathrm{can}_\pi(i) < \mathrm{can}_\pi(j)$. Since a k-track permutation does not contain any length-$k+1$ decreasing subsequence, its canonical coloring may have at most k colors (in this sense, it is an "*optimal*" coloring).

We now prove the following property, which yields a fast algorithm to compute the canonical coloring.

Proposition 4. *Let π be a permutation. The canonical coloring satisfies* $\mathrm{can}_\pi(i) = \min\{h > 0 \mid \forall j < i, \mathrm{can}_\pi(j) = h \Rightarrow \pi[j] < \pi[i]\}$.

2.5 Ordered Coloring

The canonical coloring yields some very interesting properties. However it is not stable by inheritance (*i.e.*, given an embedding, the inherited coloring of the canonical coloring is not necessarily canonical). We thus introduce a weaker form of coloring (in the sense that such colorings are not unique), called *ordered*, which do have a nice behavior with regards to embeddings, see Fig. 2(b).

Definition 8 (Ordered coloring). *A coloring c of permutation π is* ordered *if, for all $i < j$, $\pi[i] > \pi[j] \Rightarrow c(i) < c(j)$.*

Proposition 5. *The canonical coloring of a permutation π is ordered.*

Proposition 6. *Let (τ, c) be a colored permutation where c is ordered, and π be a permutation with an embedding e into τ, then the inherited coloring $c \circ e$ of π is ordered.*

Corollary 1. *If there exists an embedding of π into a k-track permutation τ, then there exists a colored embedding of (π, c) into $(\tau, \mathrm{can}_\tau)$ for some ordered k-coloring c of π.*

3 Algorithm

Given two permutations π and τ, our algorithm runs as follows. We first fix a coloring for τ, namely, the canonical coloring, using k colors. We then enumerate all ordered colorings of π with k colors (even if π would admit a k'-coloring with $k' < k$). An upper-bound on the number of such colorings is k^m. For each of these colorings, we look for a colored embedding of π into τ. If any ordered coloring yields a colored embedding, then we have found an embedding of π into τ. Otherwise, Corollary 1 ensures that no such embedding can exist.

Looking for an embedding using a given coloring is achieved (in linear time for n) by applying Rules 1 and 2 from a starting Leftmost pre-embedding, until "*no*" is returned (in which case no embedding exists) or the rules do not apply anymore (in which case the current pre-embedding is an embedding, Proposition 4). The pseudo-code of the algorithm is given in Algorithm 1 and is illustrated in Fig. 4.

```
Data: Permutations π and τ.
Result: An embedding of π into τ if it exists, and ⊥ otherwise.
Let can_τ be the canonical k-coloring of τ
forall ordered k-colorings c of π do
    Let e be the leftmost colored embedding of (π, c) into (τ, can_τ)
    while e ≠ ⊥ do
        if e contains an order (resp. value) conflict then
            /* Resolve any conflict (set e = ⊥ if this is not
               possible)                                          */
            Update e according to Rule 1 (resp. Rule 2)
        else
            /* e is an embedding of π into τ                      */
            return e
        end
    end
end
/* π does not occur in τ                                         */
return ⊥
```

Algorithm 1. Finding an occurrence of permutation π into permutation τ.

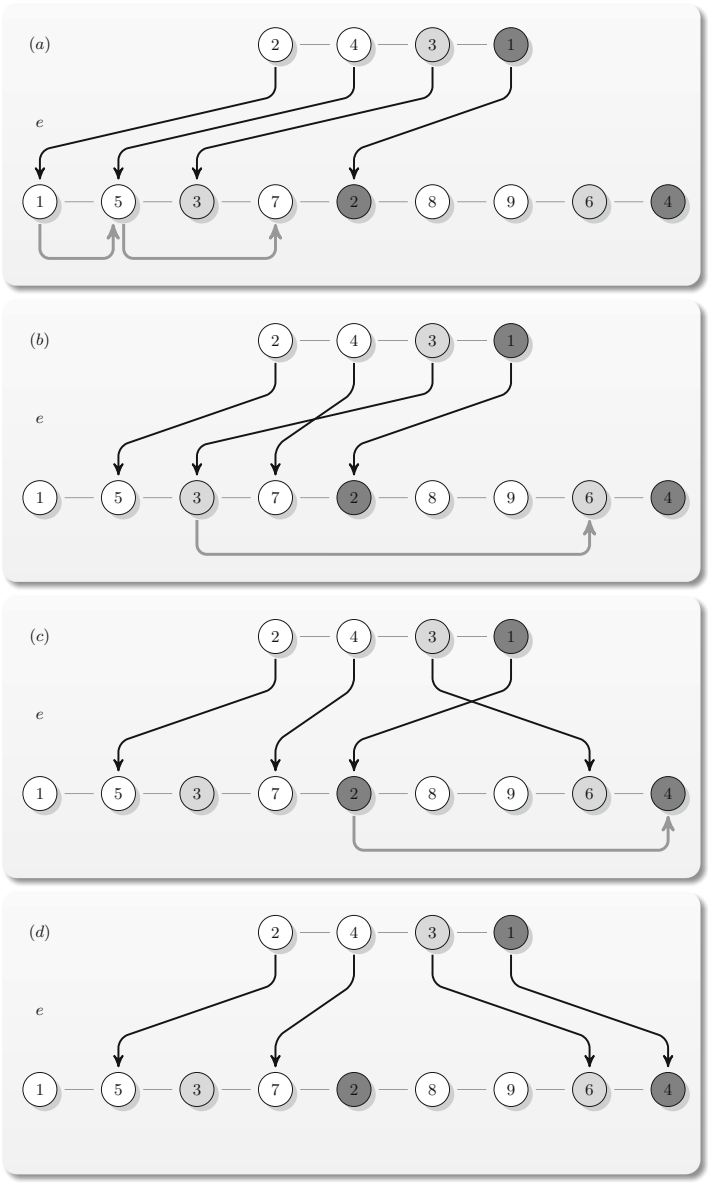


Fig. 4. Finding an order-isomorphic occurrence of $\pi = 24135$ into $\tau = 142853679$ using 3 color. Conflicts are depicted as bold links. Embedding updates (*i.e.*, resolving conflicts) are shown with horizontal bottom links. The target permutation τ is presented with the canonical 3-coloring (1) 4 (2) (8) 5 (3) 6 7 (9), whereas the source permutation π is presented with the ordered 3-coloring 2 4 (1) 3 5. **Step 1.** Initial leftmost embedding e. Identifying a value conflict for the pair $(1, 4)$, and resolve with $e(1) = 2$ and $e(2) = 4$. **Step 2.** Identifying an order conflict for the pair $(2, 3)$, and resolve with $e(3) = 8$. **Step 3.** Identifying an order conflict for the pair $(3, 4)$, and resolve with $e(4) = 9$. **Step 4.** e is both order conflict-free and value conflict-free.

4 Benchmarks

4.1 General Statements

The algorithm has been implemented in the Haskell programming language[1], and is available online at https://github.com/vialette/ppattern. All experiments were conducted on a Intel(R) Xeon(R) CPU E5–2630 v2 2.60 GHz CPU and 64GB 1866 MHz DDR3 RAM desktop computer running 4.4.38 – 1-MANJARO linux. Programs were compiled with `ghc` (GHC - Glasgow Haskell Compiler) version 7.10.3 with `-O2` option using version 1.24.0.2 of the Cabal library. For efficiently and using fewer resources, many optimizations have been implemented in the program that cannot be detailed here. At the most general level (and in setting aside pure Haskell optimization techniques), these include: considering colored embeddings of π in a tree-like manner to speed up computations, efficiently detecting and resolving (if possible) both order and value conflicts, rejecting as soon as possible wrong colored embeddings, ...

4.2 Generating Random k-Split Permutations

In this case study, we present pattern matching benchmarks for random k-track permutations with moderate parameter k. A word of caution is needed here. Indeed, efficiently generating *uniformly* at random permutations that are the union of at most k increasing subsequences remains - as far as we know - an open problem, and the accept-reject method is not at all well-suited for generating long k-track permutations for moderate parameter k. Pursuant to this, we have adopted the following procedure for random generating a permutation of size n that is the union of at most k increasing sequences. **Step 1.** Generate uniformly at random a partition of the integer n into k positive integers $n = n_1 + n_2 + \cdots + n_k$ (see for example [19]). **Step 2.** Distribute uniformly at random the positive integers $1, 2, \ldots, n$ into k increasing sequences $L_1, L_2, \ldots, L_k$ of length $n_1, n_2, \ldots, n_k$, respectively. **Step 3.** Random shuffle the k increasing sequences $L_1, L_2, \ldots, L_k$ to produce a permutation of size n that is the union of at most k increasing sequences.

For example, to generate a random permutation of length 9 that is the union of at most 3 increasing subsequences, (i) one generates uniformly at random an integer partition of 9 into 3 positive integers, say $9 = 4 + 3 + 2$, (ii) distributes uniformly at random the positive integers $1, 2, \ldots, 9$ into 3 increasing sequences of length 4, 3 and 2, say $L_1 = 1\,5\,6\,9$, $L_2 = 2\,3\,8$ and $L_3 = 4\,7$, and (iii) random shuffles L_1, L_2 and L_3, say $2\,1\,5\,3\,6\,4\,7\,8\,9$.

Figure 5 compares the exact distribution of permutations of length $n = 1, 2, \ldots, 10$ which are the union of at most k increasings patterns with the proposed k-track generation algorithm (*i.e.*, how many increasing patterns are really needed for a permutation that is the union of at most k increasing patterns?).

[1] https://www.haskell.org/.

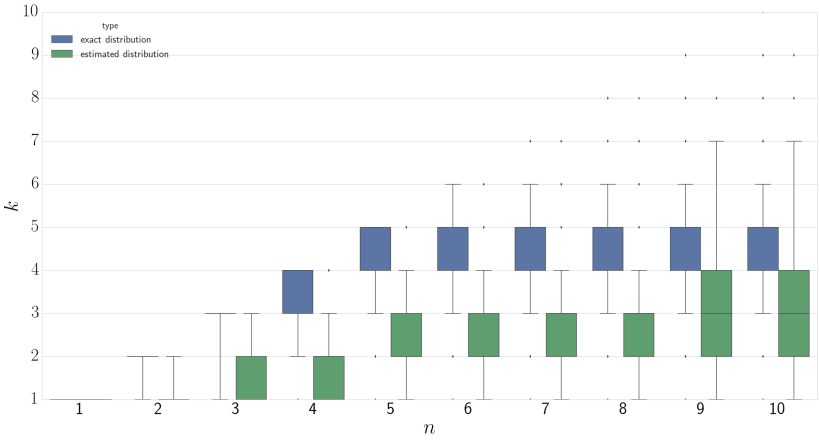


Fig. 5. Comparing the exact distribution of permutations of length n which are the union of at most k increasings patterns with the proposed k-track generation algorithm.

As it can be seen, the k-track generation algorithm tends to produce permutations that need less increasing patterns compared to the exact distribution (this is confirmed by one-sample t-tests for $n = 1, 2, \ldots, 10$).

4.3 k-Track Permutations

To test the algorithm, we benchmarked the running time of our implementation for several pattern sizes ($m = 10, 20$) and different k-track parameters ($k = 2, 3, 4, 5, 6$). All tests are for searching a random k-track pattern in a random k-track permutation. Several strategies are conceivable for resolving both order and value conflicts.

We conducted our tests using 2 different strategies: *Leftmost conflict selection first* and *Rightmost conflict selection first* (in both cases, the conflict may be an order or a value conflict). The benchmarks for permutations of length $n = 200, 400, 600$ are shown in Fig. 6. We investigated the differences between the two strategies (Leftmost conflict selection first and Rightmost conflict selection first) by conducting paired t-tests. The paired t-tests confirm a clear trend towards a better Rightmost conflict selection first strategy as most p-values are indicating that there are significant differences (at a 99% confidence level) between the running time means of each strategy. In the light of this trend, we focus on the running time of the algorithm for three conflict selection strategies: Rightmost order conflict first, Rightmost value conflict first and Rightmost conflict first. Analysis of variance (ANOVA) shows, however, no significant difference (at a 99% confidence level) between the means of each rightmost strategy.

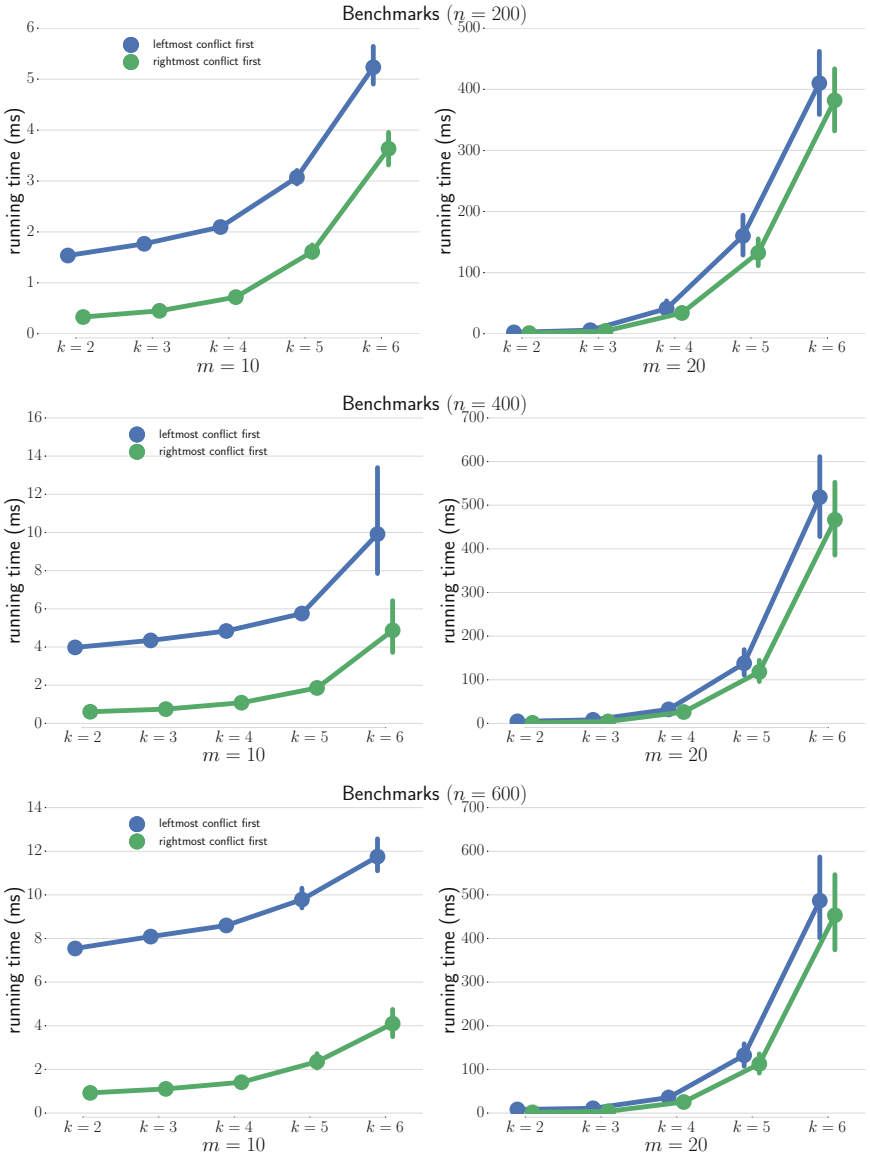


Fig. 6. Benchmarking the algorithm for $m = 10, 20$, $k = 2, 3, 4, 5, 6$, $n = 200, 400, 600$ and two conflict selection strategies: Leftmost conflict first and Rightmost conflict first.

References

1. Ahal, S., Rabinovich, Y.: On complexity of the subpattern problem. SIAM JDM **22**(2), 629–649 (2008)
2. Albert, M.H., Aldred, R.E.L., Atkinson, M.D., Holton, D.A.: Algorithms for pattern involvement in permutations. In: Eades, P., Takaoka, T. (eds.) ISAAC 2001. LNCS, vol. 2223, pp. 355–367. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45678-3_31
3. Albert, M.H., Lackner, M.-L., Lackner, M., Vatter, V.: The complexity of pattern matching for 321-avoiding and skew-merged permutations. DMTCS **18**(2) (2016)
4. Atkinson, M.D.: Permutations which are the union of an increasing and a decreasing. EJC **5**, 263–273 (1998)
5. Baik, J., Deift, P., Johansson, K.: On the distribution of the length of the longest increasing subsequence of random permutations. J. Amer. Math. Soc. **12**, 1119–1178 (1999)
6. Bose, P., Buss, J.F., Lubiw, A.: Pattern matching for permutations. IIPL **65**(5), 277–283 (1998)
7. Brandstädt, A., Kratsch, D.: On partitions of permutations into increasing and decreasing subsequences. Elektron. Inf. Verarb. Kybern. EIK **22**, 263–273 (1986)
8. Bruner, M.-L., Lackner, M.: The computational landscape of permutation patterns. CoRR, abs/1301.0340 (2013)
9. Bruner, M.-L., Lackner, M.: A fast algorithm for permutation pattern matching based on alternating runs. Algorithmica **75**(1), 84–117 (2016)
10. Crochemore, M., Porat, E.: Fast computation of a longest increasing subsequence and application. Inf. Comput. **208**(9), 1054–1059 (2010)
11. Fomin, F., Kratsch, D., Novelli, J.-C.: Approximating minimum cocolourings. IPL **84**, 285–290 (2002)
12. Guillemot, S., Marx, D.: Finding small patterns in permutations in linear time. In: Chekuri, C. (ed.) SODA, pp. 82–101. SIAM (2014)
13. Guillemot, S., Vialette, S.: Pattern matching for 321-avoiding permutations. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 1064–1073. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10631-6_107
14. Ibarra, L.: Finding pattern matchings for permutations. IPL **61**(6), 293–295 (1997)
15. Kézdy, A.E.: Partitioning permutations into increasing and decreasing subsequences. J. Comb. Theor. A **73**(2), 353–359 (1996)
16. Kitaev, S.: Patterns in Permutations and Words. Monographs in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-17333-2
17. Jelínek, V., Kynčl, J.: Hardness of permutation pattern matching. In: Klein, P. (ed.) SODA, pp. 378–396. SIAM (2017)
18. Neou, B.E., Rizzi, R., Vialette, S.: Pattern matching for separable permutations. In: Inenaga, S., Sadakane, K., Sakai, T. (eds.) SPIRE 2016. LNCS, vol. 9954, pp. 260–272. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46049-9_25
19. Nijenhuis, A., Wilf, H.S.: Combinatorial Algorithms for Computers and Calculators. Computer Science and Applied Mathematics, 2nd edn. Academic Press, New York (1978)
20. Romik, D.: The Surprising Mathematics of Longest Increasing Subsequences. Institute of Mathematical Statistics Textbooks, vol. 04. Cambridge University Press, Cambridge (2015)

21. Steele, J.M.: Variations on the monotone subsequence theme of Erdös and Szekeres. Discrete Probab. Algorithms **72**, 111–131 (1995)
22. Di Stefano, G., Krause, S., Lübbecke, M.E., Zimmermann, U.T.: On minimum k-modal partitions of permutations. JDA **6**(3), 381–392 (2008). https://doi.org/10.1016/j.jda.2008.01.002
23. Di Stefano, G., Krause, S., Lübbecke, M.E., Zimmermann, U.T.: On minimum k-modal partitions of permutations. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 374–385. Springer, Heidelberg (2006). https://doi.org/10.1007/11682462_36
24. Wagner, K.: Monotonic coverings of finite sets. Elektronische Informationsverarbeitung und Kybernetik **20**(12), 633–639 (1984)

Approximation Algorithms for the p-Hub Center Routing Problem in Parameterized Metric Graphs

Li-Hsuan Chen[1], Sun-Yuan Hsieh[2], Ling-Ju Hung[1(✉)], and Ralf Klasing[3]

[1] AROBOT Innovation CO., LTD., New Taipei City 235, Taiwan
{lihsuan.chen,lingju.hung}@arobot.info
[2] Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan 701, Taiwan
hsiehsy@mail.ncku.edu.tw
[3] CNRS, LaBRI, Université de Bordeaux, 351 Cours de la Libération, 33405 Talence cedex, France
ralf.klasing@labri.fr

Abstract. A complete weighted graph $G = (V, E, w)$ is called Δ_β-metric, for some $\beta \geq 1/2$, if G satisfies the β-triangle inequality, *i.e.,* $w(u, v) \leq \beta \cdot (w(u, x) + w(x, v))$ for all vertices $u, v, x \in V$. Given a Δ_β-metric graph $G = (V, E, w)$, the Single Allocation at most p-Hub Center Routing problem is to find a spanning subgraph H^* of G such that (i) any pair of vertices in C^* is adjacent in H^* where $C^* \subset V$ and $|C^*| \leq p$; (ii) any pair of vertices in $V \setminus C^*$ is not adjacent in H^*; (iii) each $v \in V \setminus C^*$ is adjacent to exactly one vertex in C^*; and (iv) the routing cost $r(H^*) = \sum_{u,v \in V} d_{H^*}(u, v)$ is minimized where $d_{H^*}(u, v) = w(u, f^*(u)) + w(f^*(u), f^*(v)) + w(v, f^*(v))$ and $f^*(u), f^*(v)$ are the vertices in C^* adjacent to u and v in H^*, respectively. Note that $w(v, f^*(v)) = 0$ if $v \in C^*$. The vertices selected in C^* are called *hubs* and the rest of vertices are called *non-hubs*. In this paper, we show that the Single Allocation at most p-Hub Center Routing problem is NP-hard in Δ_β-metric graphs for any $\beta > 1/2$. Moreover, we give 2β-approximation algorithms running in time $O(n^2)$ for any $\beta > 1/2$ where n is the number of vertices in the input graph.

1 Introduction

The design of hub-and-spoke networks is a key issue with applications in transportation, *e.g.,* airlines [21] and cargo delivery systems [31]. The major concern to design a hub-and-spoke network with high quality is to connect a large amount

This study has been carried out in the frame of the "Investments for the future" Programme IdEx Bordeaux - CPU (ANR-10-IDEX-03-02). Research supported by the LaBRI under the "Projets émergents" program. The main work for this article was done while Li-Hsuan Chen and Ling-Ju Hung (corresponding author) were with the National Cheng Kung University.

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 115–127, 2018.
https://doi.org/10.1007/978-3-319-94667-2_10

of origin/destination (O/D) pairs by using a small number of links. The usage of hub facilities helps to reduce the connections between all nodes. To locate p hubs in hub networks in order to route the traffic between origin/destination pairs with minimum cost is the classical hub location problem called the p-Hub Median problem [30,32]. Notice that the general p-hub median problem considers that each pair of origin/destination has different unit traffic (flow) cost. We call a hub location problem *multi-allocation*, if a demand node can be served by several hubs. If each demand node can be served by exactly one hub, the hub location problem is *single-allocation*. The p-hub median problem is NP-hard. Many linear programming-based and heuristic algorithms were proposed to solve the p-hub median problem and its variants (see the survey papers [1,14,28]).

Another hub location problem, the Single Allocation p-Hub Center problem, is to choose a fixed number p of vertices as hubs and to assign each non-hub vertex to exactly one of the chosen hubs in such a way that the maximum distance/cost between origin-destination pairs is minimized [13,31]. Unlike the p-Hub Median problem to minimize the total cost of all origin-destination pairs, the Single Allocation p-Hub Center problem is to minimize the poorest service quality. Chen *et al.* [15] proved that for any $\epsilon > 0$, it is NP-hard to approximate the Single Allocation p-Hub Center problem to a ratio $\frac{4}{3} - \epsilon$ and gave a $\frac{5}{3}$-approximation algorithm running in time $O(pn^3)$ to solve the same problem. If the input graph is Δ_β-metric, it was proved that for any $\varepsilon > 0$, to approximate the Single Allocation p-Hub Center problem to a ratio $g(\beta) - \varepsilon$ is NP-hard where $g(\beta)$ is a function of β and a series of $r(\beta)$-approximation algorithms were given in [18] where $r(\beta)$ is a function of β. The Star p-Hub Center problem is another hub location problem with min-max criterion. It is to pick p nodes as hubs among the set of demand nodes connecting with the central given hub c and to connect each of the remaining demand nodes to exactly one of the p chosen hubs such that the longest path in the tree structure network is minimized. Chen *et al.* [16] showed that for any $\epsilon > 0$, to approximate the Star p-Hub Center problem to a ratio $1.5 - \epsilon$ is NP-hard and gave a $\frac{5}{3}$-approximation algorithms for the same problem. Moreover, for input graphs satisfying β-triangle inequality, *i.e.,* $w(u, v) \leq \beta \cdot (w(u, x) + w(x, v))$ for all vertices u, v, x in the input graph $G = (V, E, w)$ and $\beta \geq 1/2$, it was shown that for any $\epsilon > 0$, to approximate the Star p-Hub Center problem to a ratio $g(\beta) - \epsilon$ is NP-hard and $r(\beta)$-approximation algorithms were given in the same paper where $g(\beta)$ and $r(\beta)$ are functions of β [17,19].

Despite numerous research results on solving various hub location problems in the past twenty-five years [14,20], the design of approximation algorithms for hub location problems only made very little progress in the past two decades, especially for the p-Hub Median problem [25,26]. In this paper, we consider a variant of the p-Hub Median problem in which each pair of origin/destination has the same unit traffic (flow) cost called the Single Allocation at most p-Hub Center Routing problem. The Single Allocation at most p-Hub Center Routing problem is to choose at most p vertices as *hubs* and to assign each remaining vertex (called *non-hub*) to exactly one of the chosen hubs in

such a way that the sum of distance/cost between all origin-destination pairs is minimized, *i.e.*, the *routing cost* is minimized. There are some routing cost optimization problems on finding a spanning subtrees or a spanning tree satisfying certain properties of the input graph such that the routing cost is minimized [27,33,34]. Some of these minimum routing cost spanning tree problems admit polynomial-time approximation schemes [33,34].

Our study uses the well-known concept of *stability of approximation* for hard optimization problems [9,11,22,23]. The idea of this concept is similar to that of the stability of numerical algorithms. But instead of observing the size of the change in the output value according to a small change of the input value, one is interested in the size of the change of the approximation ratio according to a small change in the specification (some parameters, characteristics) of the set of problem instances considered. If the change of the approximation ratio is small for every small change in the set of problem instances, then the algorithm is called *stable*. The concept of *stability of approximation* has been successfully applied to several fundamental hard optimization problems. E.g. in [2–4,8–10, 12,29] it was shown that one can partition the set of all input instances of the Traveling Salesman Problem into infinitely many subclasses according to the degree of violation of the triangle inequality, and for each subclass one can guarantee upper and lower bounds on the approximation ratio. Similar studies demonstrated that the β-triangle inequality can serve as a measure of hardness of the input instances for other problems as well, in particular for the problem of constructing 2-connected spanning subgraphs of a given complete edge-weighted graph [5], and for the problem of finding, for a given positive integer $k \geq 2$ and an edge-weighted graph G, a minimum k-edge- or k-vertex-connected spanning subgraph [6,7].

In this paper, we consider a graph $G = (V, E, w)$ with a distance function $w(\cdot, \cdot)$ being a Δ_β-metric graph on V such that $w(v, v) = 0$, $w(u, v) = w(v, u)$, and $w(u, v) \leq \beta \cdot (w(u, x) + w(x, v))$ for all $u, v, x \in V$. Given a positive integer p, let H^* be a spanning subgraph of G satisfying the conditions that vertices (*hubs*) in $C^* \subset V$ form a clique of size at most p in H^*, vertices (*non-hubs*) in $V \setminus C^*$ form an independent set in H^*, and each non-hub $v \in V \setminus C^*$ is adjacent to exactly one hub in C^*. Define $d_{H^*}(u, v) = w(u, f^*(u)) + w(f^*(u), f^*(v)) + w(v, f^*(v))$ where $f^*(u)$ and $f^*(v)$ are hubs adjacent to u and v in H^* respectively. Notice that if u is a hub in H^* then $w(u, f^*(u)) = 0$. Let $r(H^*) = \sum_{u,v \in V} d_{H^*}(u, v)$ be the *routing cost* of H^*. We list the formal definition of the Single Allocation at most p-Hub Center Routing problem in the following.

Single Allocation at most p-Hub Center Routing (Δ_β-SApHCR)

Input: A Δ_β-metric graph $G = (V, E, w)$ and a positive integer p.

Output: A spanning subgraph H^* of G satisfying the following conditions (i) any pair of vertices (hubs) in C^* is adjacent in H^* where $C^* \subset V$ and $|C^*| \leq p$; (ii) any pair of vertices (non-hubs) in $V \setminus C^*$ is not adjacent in H^*; (iii) each non-hub $v \in V \setminus C^*$ is adjacent to exactly one hub in C^* such that $r(H^*)$ is minimized.

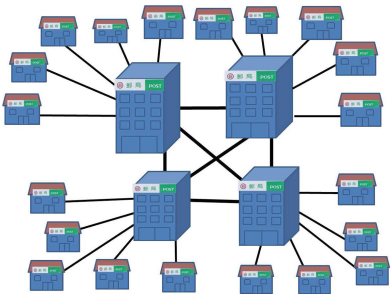


Fig. 1. An example of a single allocation at most p-hub center routing network where the four hubs are the major post offices and the non-hubs are the other small post offices.

In Fig. 1, we give an example of the Single Allocation at most p-Hub Center Routing problem that can be applied in the design of post mail networks for which hubs are major post offices and non-hubs are small post offices. In this paper, we investigate the approximability of the Single Allocation at most p-Hub Center Routing problem in Δ_β-metric graphs. The paper is organized as follows: In Sect. 2, we prove that the Single Allocation at most p-Hub Center Routing problem is NP-hard in Δ_β-metric graphs for any $\beta > 1/2$. In Sect. 3, for any $\beta > 1/2$, we give 2β-approximation algorithms running in time $O(n^2)$ for the Single Allocation at most p-Hub Center Routing problem.

2 NP-hardness

In this section, we show that for any $\beta > 1/2$, the Single Allocation at most p-Hub Center Routing problem is NP-hard.

Theorem 1. *For any $\beta > 1/2$, the* Single Allocation at most p-Hub Center Routing *problem in Δ_β-metric graphs is NP-hard.*

Proof. We prove that the Single Allocation at most p-Hub Center Routing problem is at least as hard as the well-known NP-hard problem Maximum Clique [24].

Maximum Clique Problem [24]
Input: A simple undirected graph $G = (V, E)$ and a positive integer k.
Output: Whether there is a clique $S \subseteq V$ of size k in G.

Notice that if G has a universal vertex v, *i.e.,* $deg_G(v) = |V| - 1$, then we can simply select v in S and ask whether there exists a size $k-1$ clique in $G[V \setminus \{v\}]$. Thus, we may assume that G has no universal vertex. To show such a statement, we reduce the input $G = (V, E)$ of the Maximum Clique problem to the Single Allocation at most p-Hub Center Routing problem where $p = k + 1$.

According to G, we construct an input Δ_β-metric graph $G' = (V', E', w)$ where $V' = V \cup \{x\}$, $E' = \{(u, v) \mid u, v \in V'\}$, and assign the cost of each edge in E' as follows.

- $w(u, v) = 1$ if (u, v) is an edge in G.
- $w(u, v) = 1 + \epsilon$ if (u, v) is a non-edge in G where $0 < \epsilon < 1$.
- $w(x, v) = 1$ for all $v \in V$.

It is not hard to see that G' is a Δ_β-metric graph for any $\beta \geq \frac{1+\epsilon}{2}$. Notice that for any constant β, it defines a Δ_β-metric graph class and this graph class contains all $\Delta_{\beta'}$-metric graphs for $\beta' \leq \beta$. If one can prove for any $1/2 < \beta < 1$ the Single Allocation at most p-Hub Center Routing problem is NP-hard, then it implies for any $\beta > 1/2$ this problem is NP-hard. In the following proof, we may assume that $1/2 < \beta < 1$.

Let H^* be an optimal solution of the Single Allocation at most p-Hub Center Routing problem. Let S^* be a size k clique in G. We then obtain a solution H of the Single Allocation at most p-Hub Center Routing problem by letting all vertices $C = S^* \cup \{x\}$ be the set of hubs in H and letting all the remaining vertices in $V \setminus S^*$ be non-hubs adjacent to x. We obtain the following facts.

- For two non-hubs y, z, $dist_H(y, z) = 2$.
- For a hub $v \in C \setminus \{x\}$ and a non-hub y, $dist_H(v, y) = 2$.
- For two hubs $u, v \in C$, $dist_H(u, v) = w(u, v)$.
- For any vertex $v \in V$, $dist_H(v, x) = w(v, x) = 1$.

We see that the routing cost of H is $r(H) = 2 \cdot \binom{n}{2} - \binom{k}{2} + n = n^2 - \binom{k}{2}$ where $n = |V|$.

Since H^* is an optimal solution of the Single Allocation at most p-Hub Center Routing problem in G', we have $r(H^*) \leq n^2 - \binom{k}{2}$.

Claim 1. *All non-hubs in H^* must be adjacent to the same hub.*

Proof. Suppose that there are at least two hubs adjacent to non-hubs. Then the routing cost between any two non-hubs which are adjacent to different hubs is at least 3. This will imply that $r(H^*) \geq (\frac{3-\rho}{2}) \cdot n^2 - \binom{k}{2} > n^2 - \binom{k}{2}$ where $0 < \rho < 1$, a contradiction to the assumption that H^* is an optimal solution of the Single Allocation at most p-Hub Center Routing. ■

Claim 2. *The number of hubs in H^* is p.*

Proof. If the number of hubs is less than p, *i.e.*, $|C^*| < p$, then we may obtain another solution H' by selecting $p - |C^*|$ non-hubs in H^* and let them be hubs in H'. Since all hubs are pairwise adjacent and $\beta < 1$, it is not hard to see that $r(H') < r(H^*)$. It contradicts the assumption that H^* is an optimal solution of the Single Allocation at most p-Hub Center Routing. This shows that the number of hubs in H^* must be p. ■

Claim 3. *The vertex x must be a hub in H^*.*

Proof. Suppose that x is not a hub. Since G has no universal vertex, in H^* the hub which is adjacent to all non-hubs must be incident to some edges with edge cost $1+\epsilon$. We see that $r(H^*) \geq (\frac{2+\epsilon'}{2}) \cdot n^2 - \binom{k}{2} > n^2 - \binom{k}{2} = r(H)$ where $\epsilon' > 0$, a contradiction to the assumption that H^* is an optimal solution of the Single Allocation at most p-Hub Center Routing. This completes the proof that x must be a hub in H^*. ■

Claim 4. *If the* Single Allocation at most p-Hub Center Routing *has an optimal solution H^* with $r(H^*) = n^2 - \binom{k}{2}$ and C^* is the set of hubs in H^*, then $C^* \setminus \{x\}$ is a clique of size k in G where $k = p - 1$.*

Proof. According to Claims 1–3 that all non-hubs are adjacent to x and $|C^*| = p = k + 1$, we have the routing cost between vertices in C^* is

$$\begin{aligned} r(C^*) &= r(H^*) - r(V' \setminus C^*) - r(V' \setminus C^*, C^*) \\ &= \left(n^2 - \binom{k}{2}\right) - 2 \cdot \binom{n-k}{2} - ((n-k) + 2 \cdot k(n-k)) = \binom{k+1}{2}. \end{aligned}$$

Notice that $w(x, v) = 1$ for $v \in V$, $w(u, v) = 1$ if $(u, v) \in E$, otherwise $w(u, v) = 1 + \epsilon$. Since $r(C^*) = \sum_{u,v \in C^*} w(u, v) = \binom{k+1}{2}$, we see that for $u, v \in C^* \setminus \{x\}$, $w(u, v) = 1$ and $C^* \setminus \{x\}$ forms a clique in G. ■

According to Claim 4, if there exists a polynomial time algorithm that solves Single Allocation at most p-Hub Center Routing with routing cost $n^2 - \binom{k}{2}$ where $k = p - 1$, then the Maximum Clique problem can be solved in polynomial time. However, Maximum Clique is a well-known NP-hard problem [24]. By the fact Maximum Clique is an NP-hard problem, this implies that Single Allocation at most p-Hub Center Routing is also an NP-hard problem. □

3 New Approximation Algorithms

We have shown the NP-hardness of the Single Allocation at most p-Hub Center Routing problem by reducing from the maximum clique problem. It is well-known that the maximum clique is hard to approximate. In this section, we give approximation algorithms for the Single Allocation at most p-Hub Center Routing problem.

We first state a property of Δ_β-metric graphs in the following lemma.

Lemma 1 [8]**.** *Let $G = (V, E)$ be a Δ_β-metric graph for $\frac{1}{2} \leq \beta < 1$. For any two edges $(u, x), (v, x)$ with a common endvertex x in G, $w(u, x) \leq \frac{\beta}{1-\beta} \cdot w(v, x)$.*

Algorithm 1. Approximation algorithm for Δ_β-SApHCR for $1/2 \leq \beta \leq 1$

Let $U := V$. Initially, $C = \emptyset$. Construct a spanning subgraph H of G by the following steps.

Step 1: Find $z = \arg\min_{v \in V} \sum_{u \in V} w(u, v)$ as a hub in H.
Step 2: Pick $p - 1$ vertices $\{v_1, \ldots, v_{p-1}\}$ farthest to z from U. Let $C := C \cup \{z, v_1, \ldots, v_{p-1}\}$ be the set of hubs in H and $U := U \setminus \{z, v_1, \ldots, v_{p-1}\}$.
Step 3: Connect all vertices in U to z as non-hubs in H.
Step 4: Return H.

Theorem 2. *For any $1/2 \leq \beta \leq 1$, there is a 2β-approximation algorithm for the* Single Allocation at most p-Hub Center Routing *problem.*

Proof. It is easy to see that in time $O(n^2)$, Algorithm 1 returns a feasible solution of the Single Allocation at most p-Hub Center Routing problem. We now prove that the solution H returned by Algorithm 1 satisfies the approximation ratio 2β. Let $G = (V, E, w)$ be the input graph of the Single Allocation at most p-Hub Center Routing problem. Let H^* be an optimal solution of the Single Allocation at most p-Hub Center Routing problem. Let C^* denote the set of hubs in H^*. Define $w(H^*) = \sum_{(u,v) \in E(H^*)} w(u, v)$. Construct a weighted complete graph $G^* = (V, E, w^*)$ according to H^* where $w^*(u, v) = dist_{H^*}(u, v)$. Define

$$w^*(G^*) = \sum_{u,v \in V} w^*(u, v) = \sum_{u,v \in V} dist_{H^*}(u, v) = r(H^*).$$

Let H be the solution returned by Algorithm 1 with z being the only hub in H that is adjacent to non-hubs. Let S_z be the spanning star of G with center z satisfying $z = \arg\min_{v \in V} \sum_{u \in V} w(u, v)$. We use $f^*(z)$ to denote the hub adjacent to z in H^*. Note that $f^*(z) = z$ if z is a hub in H^*. Let S_v be the spanning star of G^* with center v and $w^*(S_v) = \sum_{(u,v) \in E(S_v)} w^*(u, v)$.

Claim 1. $r(H) \leq r(S_z) - (1 - \beta) \cdot \sum_{u,v \in C \setminus \{z\}} (w(z, u) + w(z, v))$

Proof. According to the β-triangle inequality for $u, v \in V$, $w(u, v) \leq \beta(w(z, u) + w(z, v))$. We obtain that

$$\begin{aligned} r(H) &= r(S_z) - \sum_{u,v \in C \setminus \{z\}} (w(z, u) + w(z, v) - w(u, v)) \\ &\leq r(S_z) - (1 - \beta) \cdot \sum_{u,v \in C \setminus \{z\}} (w(z, u) + w(z, v)). \end{aligned}$$

This completes the proof. ∎

Claim 2. $\sum_{u,v\in C^*\setminus\{f^*(z)\}} w(u,v) \le \sum_{u,v\in C\setminus\{z\}} (w(z,u)+w(z,v))$

Proof. We obtain that

$$\begin{aligned}\sum_{u,v\in C^*\setminus\{f^*(z)\}} w(u,v) &\le \sum_{u,v\in C^*\setminus\{f^*(z)\}} \beta\cdot(w(z,u)+w(z,v))\\ &\le \sum_{u,v\in C\setminus\{z\}} \beta\cdot(w(z,u)+w(z,v))\\ &\qquad \text{(since the selection of hubs in Algorithm 1)}\\ &\le \sum_{u,v\in C\setminus\{z\}} (w(z,u)+w(z,v)).\end{aligned}$$

This completes the proof. ■

Now we prove $r(H) \le 2\beta\cdot r(H^*)$ in the following.

$$\begin{aligned}r(H^*) = w^*(G^*) &= \frac{1}{2}\cdot\sum_{v\in V} w^*(S_v)\\ &\ge \frac{1}{2\beta}\cdot\sum_{v\in V} w(S_v) - \left(\frac{1-\beta}{2\beta}\right)\cdot\sum_{(u,v)\in E(H^*)} w^*(u,v)\\ &= \frac{1}{2\beta}\cdot\sum_{v\in V} w(S_v) - \left(\frac{1-\beta}{2\beta}\right)\cdot\sum_{(u,v)\in E(H^*)} w(u,v)\\ &= \frac{1}{2\beta}\cdot\sum_{v\in V} w(S_v) - \left(\frac{1-\beta}{2\beta}\right)\cdot\\ &\quad\left(\sum_{u\in V\setminus C^*} w(u,f^*(u)) + \sum_{u\in C^*\setminus\{f^*(z)\}} w(u,f^*(z)) + \sum_{u,v\in C^*\setminus\{f^*(z)\}} w(u,v)\right)\\ &\ge \frac{1}{2\beta}\cdot\sum_{v\in V} w(S_v) - \frac{1}{2}\cdot\left(\sum_{u\in V\setminus C^*} w(u,z) + \sum_{u\in C^*\setminus\{f^*(z)\}} w(u,z)\right) -\\ &\quad\left(\frac{1-\beta}{2\beta}\right)\cdot\sum_{u,v\in C^*\setminus\{f^*(z)\}} w(u,v)\\ &\qquad \text{(by Lemma 1, } w(u,f^*(u)) \le \tfrac{\beta}{1-\beta}\cdot w(u,z))\\ &= \frac{1}{2\beta}\cdot\sum_{v\in V} w(S_v) - \frac{1}{2}\cdot w(S_z) - \left(\frac{1-\beta}{2\beta}\right)\cdot\sum_{u,v\in C^*\setminus\{f^*(z)\}} w(u,v)\\ &\ge \left(\frac{n}{2\beta}\right)\cdot w(S_z) - \frac{1}{2\beta}\cdot w(S_z) - \left(\frac{1-\beta}{2\beta}\right)\cdot\sum_{u,v\in C^*\setminus\{f^*(z)\}} w(u,v)\\ &\qquad \text{(since } \beta\le 1)\end{aligned}$$

Algorithm 2. Approximation algorithm for Δ_β-SApHCR for $\beta \geq 1$

Let $U := V$. Initially, $C = \emptyset$. Construct a spanning subgraph H of G by the following steps.

Step 1: Find $z = \arg\min_{v \in V} \sum_{u \in V} w(u, v)$ as the hub in H.
Step 2: Connect all vertices in $U \setminus \{z\}$ to z as non-hubs in H.
Step 3: Return H.

$$
\begin{aligned}
&= \left(\frac{n-1}{2\beta}\right) \cdot w(S_z) - \left(\frac{1-\beta}{2\beta}\right) \cdot \sum_{u,v \in C^* \setminus \{f^*(z)\}} w(u, v) \\
&= \frac{r(S_z)}{2\beta} - \left(\frac{1-\beta}{2\beta}\right) \cdot \sum_{u,v \in C^* \setminus \{f^*(z)\}} w(u, v) \\
&\quad (\text{since } r(S_z) = (n-1) \cdot w(S_z)) \\
&\geq \frac{r(S_z)}{2\beta} - \left(\frac{1-\beta}{2\beta}\right) \cdot \sum_{u,v \in C \setminus \{z\}} (w(z, u) + w(z, v)) \quad (\text{by Claim 2}) \\
&\geq \frac{r(H)}{2\beta} \quad (\text{by Claim 1}).
\end{aligned}
$$

This shows that $r(H) \leq 2\beta \cdot r(H^*)$, and the proof is completed. □

Theorem 3. *For any $\beta \geq 1$, there is a 2β-approximation algorithm for the* Single Allocation at most p-Hub Center Routing *problem.*

Proof. It is easy to see that in time $O(n^2)$, Algorithm 2 returns a feasible solution of the Single Allocation at most p-Hub Center Routing problem. We now prove that the solution H returned by Algorithm 2 satisfies the approximation ratio 2β. Let $G = (V, E, w)$ be the input graph of the Single Allocation at most p-Hub Center Routing problem. Let H^* be an optimal solution of the Single Allocation at most p-Hub Center Routing problem and C^* be the set of hubs in H^*. Construct a weighted complete graph $G^* = (V, E, w^*)$ according to H^* where $w^*(u, v) = dist_{H^*}(u, v)$. Let $w(G^*) = \sum_{u,v} w^*(u, v) = r(H^*)$.

We use S_v to denote the spanning star of G with center v and $w(S_v) = \sum_{u \in V} w(u, v)$. Let $x = \arg\min_{v \in C^*}\{w(S_v)\}$. Define $w^*(S_v) = \sum_{u \in V} w^*(u, v)$ where $w^*(u, v) = dist_{H^*}(u, v)$. Let $f^*(x)$ denote the hub adjacent to x in H^*. Note that if x is a hub in H^*, then $f^*(x) = x$.

Claim 1. $w^*(S_x) = (n-2) \cdot w(x, f^*(x)) + w^*(S_{f^*(x)})$.

Proof. If x is a hub in H^*, we have $f^*(x) = x$ and the equation holds directly. Suppose that x is not a hub. We obtain that

$$\begin{aligned} w^*(S_x) &= \sum_{v \in V} w^*(x, v) = \sum_{v \in V \setminus \{x\}} w(x, f^*(x)) + w(f^*(x), f^*(v)) + w(f^*(v), v) \\ &= (n-1) \cdot w(x, f^*(x)) + \left(\sum_{v \in V \setminus \{x\}} w(f^*(x), f^*(v)) + w(f^*(v), v) \right) \\ &= (n-2) \cdot w(x, f^*(x)) + \left(\sum_{v \in V} w(f^*(x), f^*(v)) + w(f^*(v), v) \right) \\ &= (n-2) \cdot w(x, f^*(x)) + \left(\sum_{v \in V} w^*(f^*(x), v) \right) \\ &= (n-2) \cdot w(x, f^*(x)) + w^*(S_{f^*(x)}). \end{aligned}$$

This completes the proof. ■

Claim 2. For any hub $y \in C^*$, $w^*(S_y) \geq \frac{1}{\beta} \cdot w(S_y)$.

Proof. According to the β-triangle inequality, we see that $w(u, y) \leq \beta \cdot (w(u, y) + w(f^*(u), y))$. We obtain that for $u \in V$, $w^*(u, y) = w(u, f^*(u)) + w(f^*(u), y) \geq \frac{1}{\beta} \cdot w(u, y)$. Thus

$$w^*(S_y) = \sum_{u \in V} w^*(u, y) \geq \sum_{u \in V} \frac{1}{\beta} \cdot w(u, y) = \frac{1}{\beta} \cdot \sum_{u \in V} w(u, y) = \frac{1}{\beta} \cdot w(S_y).$$

This completes the proof. ■

Now we prove $r(H) \leq 2\beta \cdot r(H^*)$ in the following.

$$\begin{aligned} r(H^*) = w^*(G^*) &= \frac{1}{2} \cdot \sum_{v \in V} w^*(S_v) \\ &= \frac{1}{2} \cdot \left(\sum_{v \in V} \left(w^*(S_{f^*(v)}) + (n-2) \cdot w(v, f^*(v)) \right) \right) \qquad \text{(by Claim 1)} \\ &\geq \frac{1}{2} \cdot \sum_{v \in V} w^*(S_{f^*(v)}) \geq \frac{1}{2\beta} \cdot \sum_{v \in V} w(S_{f^*(v)}) \quad \text{(by Claim 2)} \\ &\geq \frac{1}{2\beta} \cdot n \cdot w(S_x) \quad (\text{since } x = \arg\min_{v \in C^*} \{w(S_v)\}) \\ &\geq \frac{1}{2\beta} \cdot (n-1) \cdot w(S_z) \quad (\text{since } z = \arg\min_{v \in V} \{w(S_v)\}) = \frac{1}{2\beta} \cdot r(H). \end{aligned}$$

This shows $r(H) \leq 2\beta \cdot r(H^*)$. Thus Algorithm 2 returns a solution with approximation ratio 2β. □

4 Concluding Remarks

In this paper, we have proved that the Single Allocation at most p-Hub Center Routing problem is NP-hard in Δ_β-metric graphs for any $\beta > \frac{1}{2}$. For any $\beta > \frac{1}{2}$, we have given 2β-approximation algorithms. In future work, it is of interest to design approximation algorithms with better approximation ratios. Besides, it is still open whether the Single Allocation at most p-Hub Center Routing problem is APX-hard or not. If the Single Allocation at most p-Hub Center Routing problem is APX-hard, one must prove that for any $\epsilon > 0$, it is NP-hard to approximate Single Allocation at most p-Hub Center Routing to a factor $c - \epsilon$ for some constant $c > 1$. The other possibility is that there exists a polynomial-time approximation scheme (PTAS) for Single Allocation at most p-Hub Center Routing. We conjecture that there exists a PTAS for the Single Allocation at most p-Hub Center Routing problem.

References

1. Alumur, S., Kara, B.Y.: Network hub location problems: the state of the art. Eur. J. Oper. Res. **190**, 1–21 (2008)
2. Andreae, T.: On the traveling salesman problem restricted to inputs satisfying a relaxed triangle inequality. Networks **38**, 59–67 (2001)
3. Andreae, T., Bandelt, H.-J.: Performance guarantees for approximation algorithms depending on parameterized triangle inequalities. SIAM J. Discrete Math. **8**, 1–16 (1995)
4. Bender, M.A., Chekuri, C.: Performance guarantees for the TSP with a parameterized triangle inequality. Inf. Process. Lett. **73**, 17–21 (2000)
5. Böckenhauer, H.-J., Bongartz, D., Hromkovič, J., Klasing, R., Proietti, G., Seibert, S., Unger, W.: On the hardness of constructing minimal 2-connected spanning subgraphs in complete graphs with sharpened triangle inequality. In: Agrawal, M., Seth, A. (eds.) FSTTCS 2002. LNCS, vol. 2556, pp. 59–70. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36206-1_7
6. Böckenhauer, H.-J., Bongartz, D., Hromkovič, J., Klasing, R., Proietti, G., Seibert, S., Unger, W.: On k-edge-connectivity problems with sharpened triangle inequality. In: Petreschi, R., Persiano, G., Silvestri, R. (eds.) CIAC 2003. LNCS, vol. 2653, pp. 189–200. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-44849-7_24
7. Böckenhauer, H.-J., Bongartz, D., Hromkovič, J., Klasing, R., Proietti, G., Seibert, S., Unger, W.: On k-connectivity problems with sharpened triangle inequality. J. Discrete Algorithms **6**, 605–617 (2008)
8. Böckenhauer, H.-J., Hromkovič, J., Klasing, R., Seibert, S., Unger, W.: Approximation algorithms for the TSP with sharpened triangle inequality. Inf. Process. Lett. **75**, 133–138 (2000)
9. Böckenhauer, H.-J., Hromkovič, J., Klasing, R., Seibert, S., Unger, W.: Towards the notion of stability of approximation for hard optimization tasks and the Traveling Salesman Problem. In: Bongiovanni, G., Petreschi, R., Gambosi, G. (eds.) CIAC 2000. LNCS, vol. 1767, pp. 72–86. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-46521-9_7

10. Böckenhauer, H.-J., Hromkovič, J., Klasing, R., Seibert, S., Unger, W.: An improved lower bound on the approximability of metric TSP and approximation algorithms for the TSP with sharpened triangle inequality. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 382–394. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-46541-3_32
11. Böckenhauer, H.-J., Hromkovič, J., Seibert, S.: Stability of approximation. In: Gonzalez, T.F. (ed.) Handbook of Approximation Algorithms and Metaheuristics, chap. 31. Chapman & Hall/CRC (2007)
12. Böckenhauer, H.-J., Seibert, S.: Improved lower bounds on the approximability of the traveling salesman problem. RAIRO - Theoret. Inf. Appl. **34**, 213–255 (2000)
13. Campbell, J.F.: Integer programming formulations of discrete hub location problems. Eur. J. Oper. Res. **72**, 387–405 (1994)
14. Campbell, J.F., O'Kelly, M.E.: Twenty-five years of hub location research. Transp. Sci. **46**, 153–169 (2012)
15. Chen, L.-H., Cheng, D.-W., Hsieh, S.-Y., Hung, L.-J., Lee, C.-W., Wu, B.Y.: Approximation algorithms for single allocation k-hub center problem. In: Proceedings of the 33rd Workshop on Combinatorial Mathematics and Computation Theory (CMCT 2016), pp. 13–18 (2016)
16. Chen, L.-H., Cheng, D.-W., Hsieh, S.-Y., Hung, L.-J., Lee, C.-W., Wu, B.Y.: Approximation algorithms for the star k-hub center problem in metric graphs. In: Dinh, T.N., Thai, M.T. (eds.) COCOON 2016. LNCS, vol. 9797, pp. 222–234. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42634-1_18
17. Chen, L.-H., Hsieh, S.-Y., Hung, L.-J., Klasing, R., Lee, C.-W., Wu, B.Y.: On the complexity of the star p-hub center problem with parameterized triangle inequality. In: Fotakis, D., Pagourtzis, A., Paschos, V.T. (eds.) CIAC 2017. LNCS, vol. 10236, pp. 152–163. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57586-5_14
18. Chen, L.-H., Hsieh, S.-Y., Hung, L.-J., Klasing, R.: The approximability of the p-hub center problem with parameterized triangle inequality. In: Cao, Y., Chen, J. (eds.) COCOON 2017. LNCS, vol. 10392, pp. 112–123. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62389-4_10
19. Chen, L.-H., Cheng, D.-W., Hsieh, S.-Y., Hung, L.-J., Klasing, R., Lee, C.-W., Wu, B.Y.: Approximability and inapproximability of the star p-hub center problem with parameterized triangle inequality. J. Comput. Syst. Sci. **92**, 92–112 (2018)
20. Contreras, I.: Hub location problems. In: Laporte, G., Nickel, S., da Gama, F.S. (eds.) Location Science, pp. 311–344. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-13111-5_12
21. Ernst, A.T., Hamacher, H., Jiang, H., Krishnamoorthy, M., Woeginger, G.: Uncapacitated single and multiple allocation p-hub center problems. Comput. Oper. Res. **36**, 2230–2241 (2009)
22. Hromkovič, J.: Stability of approximation algorithms and the knapsack problem. In: Karhumäki, J., Maurer, H., Paun, G., Rozenberg, G. (eds.) Jewels are Forever, pp. 238–249. Springer, Heidelberg (1999). https://doi.org/10.1007/978-3-642-60207-8_21
23. Hromkovič, J.: Algorithmics for Hard Problems - Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics, 2nd edn. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-662-05269-3
24. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, San Francisco (1979)

25. Iwasa, M., Saito, H., Matsui, T.: Approximation algorithms for the single allocation problem in hub-and-spoke networks and related metric labeling problems. Discrete Appl. Math. **157**, 2078–2088 (2009)
26. Kuroki, Y., Matsui, T.: Approximation algorithms for hub location problems. In: The 9th Annual Meeting of Asian Association for Algorithms and Computation (AAAC 2016) (2016)
27. Lin, C.-W., Wu, B.Y.: On the minimum routing cost clustered tree problem. J. Combinat. Optim. **33**, 1106–1121 (2017)
28. Mladenović, N., Brimberg, J., Hansen, P., Moreno-Pérez, J.A.: The p-median problem: a survey of metaheuristic approaches. Eur. J. Oper. Res. **179**, 927–939 (2007)
29. Mömke, T.: An improved approximation algorithm for the traveling salesman problem with relaxed triangle inequality. Inf. Process. Lett. **115**, 866–871 (2015)
30. O'Kelly, M.E.: A quadratic integer program for the location of interacting hub facilities. Eur. J. Oper. Res. **32**, 393–404 (1987)
31. O'Kelly, M.E., Miller, H.J.: Solution strategies for the single facility minimax hub location problem. Pap. Reg. Sci. **70**, 367–380 (1991)
32. Todosijević, R., Urošević, D., Mladenović, N., Hanafi, S.: A general variable neighborhood search for solving the uncapacitated r-allocation p-hub median problem. Optim. Lett. **11**, 1109–1121 (2017)
33. Wu, B.Y., Lancia, G., Bafna, V., Chao, K.-M., Ravi, R., Tang, C.Y.: A polynomial-time approximation scheme for minimum routing cost spanning trees. SIAM J. Comput. **29**, 761–778 (1999)
34. Wu, B.Y.: A polynomial time approximation scheme for the two-source minimum routing cost spanning trees. J. Algorithms **44**, 359–378 (2002)

On the Area Requirements of Straight-Line Orthogonal Drawings of Ternary Trees

Barbara Covella, Fabrizio Frati(✉), and Maurizio Patrignani

Roma Tre University, Rome, Italy
{covella,frati,patrigna}@dia.uniroma3.it

Abstract. We prove that every n-node ternary tree has a planar straight-line orthogonal drawing in $O(n^{1.576})$ area, improving upon the previously best known $O(n^{1.631})$ bound. Further, we present an upper bound, the outcomes of an experimental evaluation, and a conjecture on the area requirements of planar straight-line orthogonal drawings of complete ternary trees.

1 Introduction

A *planar straight-line orthogonal* drawing of a graph represents each vertex as a point in the plane and each edge either as a horizontal or as a vertical straight-line segment, so that no two edges cross. Planar straight-line orthogonal graph drawings have long been studied. In 1987 Tamassia [15] presented an algorithm that decides in polynomial time whether a graph with a fixed combinatorial embedding has a planar straight-line orthogonal drawing (and, more in general, a planar orthogonal drawing with at most k bends, for any integer $k \geq 0$); this result lies at the very foundations of the research area now called Graph Drawing. Nomura et al. [10] proved that every *outerplanar graph* with maximum degree 3 and no 3-cycle has a planar straight-line orthogonal drawing.

The question whether a given *tree* has a planar straight-line orthogonal drawing is less interesting, as the answer is positive if and only if the degree of each node is at most 4. Most research efforts concerning planar straight-line orthogonal drawings of trees have then been devoted to the construction of drawings with *small area*. This is usually formalized by requiring nodes to lie at *grid* points, i.e., points with integer coordinates, and by defining the *width* and *height* of a drawing as the number of grid columns and rows intersecting it, respectively, and the *area* as the width times the height.

It has been known since the 70's that n-node complete binary trees[1] admit planar straight-line orthogonal drawings in $O(n)$ area [4,13]. Concerning general binary trees, the long-standing $O(n \log \log n)$ area bound [3,14] has recently

Partially supported by MIUR Project "MODE" under PRIN 20157EFM5C and by H2020-MSCA-RISE project 734922 – "CONNECT".

[1] Drawing algorithms usually assume trees to be *rooted*, that is, to have a distinguished node, called *root*. Trees of maximum degree 3 and 4 are then called *binary* and

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 128–140, 2018.
https://doi.org/10.1007/978-3-319-94667-2_11

been improved by Chan [2] to $n2^{O(\log^* n)}$, where $\log^*$ denotes the iterated logarithm. Worse bounds are known for ternary trees; Frati proved [7] that ternary trees and complete ternary trees admit planar straight-line orthogonal drawings in $O(n^{1.631})$ and $O(n^{1.262})$ area, respectively. The latter bound was improved to $O(n^{1.118})$ by Ali [1]. No super-linear area lower bound is known for planar straight-line orthogonal drawings of ternary trees. We prove the following result.

Theorem 1. *Every n-node ternary tree admits a planar straight-line orthogonal drawing in $O(n^{1.576})$ area.*

The proof of Theorem 1, which is presented in Sect. 2, exploits a geometric construction which includes the one presented by Frati [7] as a special case.

In Sect. 3 we study the area requirements of planar straight-line orthogonal drawings of complete ternary trees. We focus on drawings that satisfy the *subtree separation property*: the smallest axis-parallel rectangles enclosing the drawings of any two node-disjoint subtrees do not overlap. We prove that n-node complete ternary trees admit such drawings in $O(n^{1.149})$ area. We also present an algorithm that constructs a minimum-area planar straight-line orthogonal drawing with the subtree separation property of a complete ternary tree in polynomial time. This allowed us to experimentally compute the area required by planar straight-line orthogonal drawings with the subtree separation property for complete ternary trees with up to 2 billion nodes. The outcomes of these experiments led us to conjecture that complete ternary trees do not admit planar straight-line orthogonal drawings with the subtree separation property in near-linear area. In the following a *drawing* always means a planar straight-line orthogonal drawing.

2 General Ternary Trees

In this section we prove Theorem 1; we show an inductive algorithm that takes in input an n-node ternary tree T and constructs a drawing Γ of T in $O(n^{1.576})$ area that satisfies the *top-visibility property*, i.e., the vertical half-line emanating from the root $r(T)$ and directed upwards does not intersect Γ, except at $r(T)$.

For a node v in T, we denote as T_v the subtree of T rooted at v; further, we denote the subtrees of v as the *heaviest* subtree H_v, the *second heaviest* subtree M_v, and the *lightest* subtree L_v of v, according to the non-increasing order of the number of their nodes, with ties broken arbitrarily. A *heavy path* in T is a path $(v_1, \ldots, v_k)$ such that $r(T) = v_1$ and v_{i+1} is the root of H_{v_i}, for any $i = 1, \ldots, k-1$. We denote by $(\pi_1, \ldots, \pi_{k(\pi)})$ the nodes of a heavy path π.

In the base case $n = 1$. Then Γ is constructed by placing $r(T)$ at any grid point of the plane. If $n > 1$, then let π be a heavy path in T. Further, let ρ be a heavy path in $M_{r(T)}$. Let $p > 4$ be a parameter to be determined later and let x be the smallest index such that π_x has at least two subtrees with at least

ternary trees, respectively. We will use standard terminology on rooted trees, like *child*, *subtree*, and *leaf*; refer to [5,6,11]. A tree is *complete* if every non-leaf node has the same number of children and every root-to-leaf path has the same length.

n/p nodes each, if any such index exists. We first describe our construction by assuming that x exists and is greater than 2; we will deal with the other cases later. Let σ be a heavy path in $M_{\pi_{x-1}}$ and let τ be a heavy path in M_{π_x}. Let $P = (\rho_{k(\rho)}, \ldots, \rho_1, \pi_1, \ldots, \pi_{x-1}, \sigma_1, \ldots, \sigma_{k(\sigma)})$ and $Q = (\pi_{k(\pi)}, \ldots, \pi_x, \tau_1, \ldots, \tau_{k(\tau)})$.

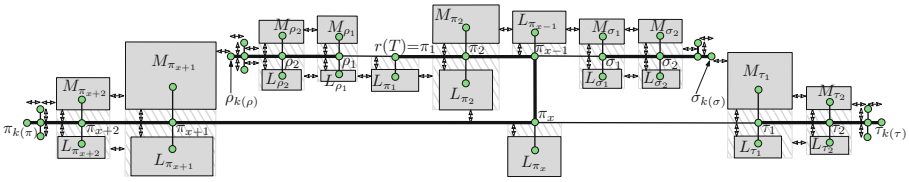


Fig. 1. Construction of Γ if $n > 1$. Fat lines represent π, ρ, σ, and τ. Double-headed arrows indicate unit distances. Gray boxes represent inductively constructed drawings.

Figure 1 shows the construction of Γ. The paths P and Q lie on two horizontal lines ℓ_P and ℓ_Q, with ℓ_P above ℓ_Q and with the nodes in left-to-right order as they appear in P and Q. Let $\mathcal{V} = V(P) \cup V(Q)$. For every subtree T^* of T rooted at a node $r^* \notin \mathcal{V}$ child of a node $p^* \in \mathcal{V}$, a drawing Γ^* of T^* is constructed inductively and attached to p^* as follows. If $T^* = L_{p^*}$, then Γ^* is placed with r^* on the same vertical line as p^* with its top side one unit below p^* (we call T^* a *bottom subtree* of P or Q, depending on whether p^* is in $V(P)$ or $V(Q)$, respectively). Otherwise, $T^* = M_{p^*}$ (note that $T^* \neq H_{p^*}$, as $r^* \notin \mathcal{V}$); then Γ^* is rotated by 180° and placed with r^* on the same vertical line as p^* with its bottom side one unit above p^* (we call T^* a *top subtree* of P or Q, depending on whether p^* is in $V(P)$ or $V(Q)$, respectively). There is one exception to this rule, which happens if $T^* = L_{p^*}$ with $p^* = \pi_{x-1}$; then Γ^* is rotated by 180° and placed with r^* on the same vertical line as p^* and with its bottom side one unit above p^*, as if it were a second heaviest subtree (we call T^* a *top subtree* of P).

The horizontal distance of the nodes in $\mathcal{V}$ is set so that, for any two nodes x and y such that x comes right before y on P or Q, the rightmost vertical line intersecting x or its attached subtrees is one unit to the left of the leftmost vertical line intersecting y or its attached subtrees. There are two exceptions to this rule, involving the distance between π_x and its neighbors π_{x+1} and τ_1. Indeed, the distance between π_x and π_{x+1} is set so that the rightmost vertical line intersecting π_{x+1} or its attached subtrees is one unit to the left of the leftmost vertical line intersecting P or its attached subtrees, or π_x, or L_{π_x}; the distance between π_x and τ_1 is set similarly. The reason for "pushing" π_{x+1} (τ_1) and its attached subtrees to the left (right) of P and its attached subtrees is to allow for a vertical compaction of Γ. In fact, the vertical distance between P and Q can be set so that the bottommost horizontal line intersecting P or its attached subtrees is one unit above Q. This completes the construction of Γ.

It is easy to see that Γ is a planar straight-line orthogonal drawing with the top-visibility property. Every grid column intersecting Γ contains at least one node of T, hence the width of Γ is in $O(n)$. We now analyze the height of Γ.

Let $H(n)$ be the maximum height of a drawing of an n-node tree constructed by the described algorithm. Then the height of Γ is at most $H(n)$. Note that $H(1) = 1$.

Let a (b) be the maximum cardinality of a top (resp. bottom) subtree of P. Let r (s) be the maximum cardinality of a top (resp. bottom) subtree of Q. By definition of x we have $|V(M_{\pi_i})|, |V(L_{\pi_i})| < n/p$, for $i \in \{1, \ldots, x-1\}$. Hence:

$$a, b < n/p. \tag{1}$$

Since the lightest subtree of the root of a tree with m nodes has less than $m/3$ nodes, we have

$$s \leq (n - a - b)/3. \tag{2}$$

We also have the following inequality, whose proof needs some case analysis.

$$r + s \leq \frac{2(p-1)}{3p} n. \tag{3}$$

Proof: Let R and S be a top and bottom subtree of Q, respectively, with $|R| = r$ and $|S| = s$. By construction $R = M_{\pi_i}$, for some $x < i < k(\pi)$, or $R = M_{\tau_i}$, for some $1 \leq i < k(\tau)$. Further, $S = L_{\pi_j}$, for some $x \leq j < k(\pi)$, or $S = L_{\tau_j}$, for some $1 \leq j < k(\tau)$. We first assume that $R = M_{\pi_i}$, for some $x < i < k(\pi)$. We distinguish five cases.

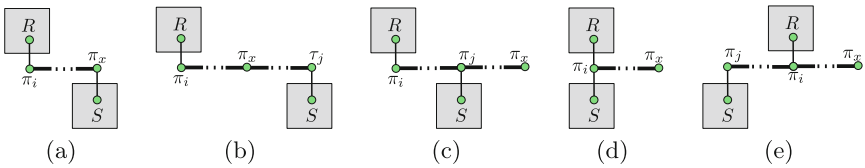


Fig. 2. The five cases in the proof of Inequality (3).

Case 1: $S = L_{\pi_x}$ (see Fig. 2(a)). We have that $|H_{\pi_i}| \geq r$; also, $|M_{\pi_x}| \geq s$. Since L_{π_x}, M_{π_x}, M_{π_i}, and H_{π_i} are vertex-disjoint, we have $2r + 2s \leq n$, which implies $r + s \leq \frac{2(p-1)}{3p} n$ if $p \geq 4$.

Case 2: $S = L_{\tau_j}$, for some $1 \leq j < k(\tau)$ (see Fig. 2(b)). We have $|H_{\pi_i}| \geq r$; also, $|H_{\tau_j}| \geq s$. Since L_{τ_j}, H_{τ_j}, M_{π_i}, and H_{π_i} are vertex-disjoint, we have $2r + 2s \leq n$, which implies $r + s \leq \frac{2(p-1)}{3p} n$ if $p \geq 4$.

Case 3: $S = L_{\pi_j}$, for some $x < j < i < k(\pi)$ (see Fig. 2(c)). We have $|H_{\pi_i}| \geq r$; also, $|M_{\pi_j}| \geq s$. Since L_{π_j}, M_{π_j}, M_{π_i}, and H_{π_i} are vertex-disjoint, we have $2r + 2s \leq n$, which implies $r + s \leq \frac{2(p-1)}{3p} n$ if $p \geq 4$.

Case 4: $S = L_{\pi_i}$ (see Fig. 2(d)). By definition of x we have $|M_{\pi_x}| \geq n/p$. Since T_{π_i} and M_{π_x} are vertex-disjoint, we have $|T_{\pi_i}| \leq \frac{p-1}{p} n$. Since $|H_{\pi_i}| \geq |M_{\pi_i}|, |L_{\pi_i}|$, we have $r + s \leq \frac{2(p-1)}{3p} n$.

Case 5: $S = L_{\pi_j}$, for some $x < i < j < k(\pi)$ (see Fig. 2(e)). As in Case 4, we have $|T_{\pi_i}| \leq \frac{p-1}{p}n$. Since $T_{\pi_{i+1}} = H_{\pi_i}$, we have $r \leq |T_{\pi_{i+1}}|$. Since $|H_{\pi_j}|, |M_{\pi_j}| \geq |L_{\pi_j}|$, we have $s \leq |T_{\pi_{i+1}}|/3$, hence $r + s \leq 4|T_{\pi_{i+1}}|/3$. On the other hand, at least $|T_{\pi_{i+1}}| - |L_{\pi_j}| \geq 2|T_{\pi_{i+1}}|/3$ nodes of $T_{\pi_{i+1}}$ do not belong to R or S, hence $\frac{4|T_{\pi_{i+1}}|}{3} + \frac{2|T_{\pi_{i+1}}|}{3} = 2|T_{\pi_{i+1}}| \leq \frac{p-1}{p}n$, which is $|T_{\pi_{i+1}}| \leq \frac{p-1}{2p}n$. It follows that $r + s \leq \frac{2(p-1)}{3p}n$.

This concludes the discussion if $R = M_{\pi_i}$, for some $x < i < k(\pi)$. Our arguments above do not make use of the fact that $|T_{\pi_{x+1}}| \geq |T_{\tau_1}|$. Hence, symmetric arguments handle the case in which $R = M_{\tau_i}$, for some $1 \leq i < k(\tau)$. □

The height of the part of Γ below ℓ_Q is given by the height of a bottom subtree of Q. Further, since ℓ_Q is one unit below the bottommost horizontal line intersecting P or its attached subtrees, the height of the part of Γ above ℓ_Q is given by the maximum between the height of a top subtree of Q, and the height of a top subtree of P plus the height of a bottom subtree of P plus one (corresponding to ℓ_P). Since the heights of a top subtree of P, a bottom subtree of P, a top subtree of Q, and a bottom subtree of Q are at most $H(a)$, $H(b)$, $H(r)$, and $H(s)$, respectively, by taking into account the grid row of ℓ_Q we get:

$$H(n) \leq \max\{H(r) + H(s) + 1, H(a) + H(b) + H(s) + 2\}. \tag{4}$$

We are going to inductively prove that

$$H(n) \leq 2 \cdot n^c - 1, \text{ where } c = \frac{1}{\log_2 \frac{3p}{p-1}}. \tag{5}$$

Note that inequality (5) is trivially true if $n = 1$. Now assume that inequality (5) is true for all integer values of the variable less than n. By inequality (4), it suffices to prove that $\max\{H(r)+H(s)+1, H(a)+H(b)+H(s)+2\} \leq 2 \cdot n^c - 1$.

– First, we need to have $H(r) + H(s) + 1 \leq 2 \cdot n^c - 1$. By induction, $H(r) + H(s) + 1 \leq 2 \cdot r^c - 1 + 2 \cdot s^c - 1 + 1$, hence we need that $r^c + s^c \leq n^c$.
Here we use Hölder's inequality, which states that $\sum a_i b_i \leq (\sum a_i^x)^{\frac{1}{x}} (\sum b_i^y)^{\frac{1}{y}}$ for every x and y such that $1/x + 1/y = 1$. By employing the values $1/x = c$, $1/y = 1 - c$, $a_1 = r^c$, $a_2 = s^c$, $b_1 = b_2 = 1$, we get:

$$r^c + s^c \leq (r + s)^c \cdot 2^{1-c} \leq \left(\frac{2(p-1)}{3p}n\right)^c \cdot 2^{1-c} = 2 \cdot \left(\frac{p-1}{3p}n\right)^c,$$

where we exploited inequality (3). Thus, we need $2 \cdot (\frac{p-1}{3p}n)^c \leq n^c$, which is $2 \cdot (\frac{p-1}{3p})^{1/\left(\log_2 \frac{3p}{p-1}\right)} \leq 1$. Set $x = \frac{3p}{p-1}$; then the previous inequality becomes $(1/x)^{1/\log_2 x} \leq 1/2$; taking the base-2 logarithms, we have $\log_2(1/x)^{1/\log_2 x} \leq \log_2(1/2)$, hence $\frac{1}{\log_2 x} \log_2(1/x) \leq -1$ which is $-1 \leq -1$. This proves that $H(r) + H(s) + 1 \leq 2 \cdot n^c - 1$.

– Second, we need to have $H(a)+H(b)+H(s)+2 \leq 2 \cdot n^c - 1$. By inequality (2), we have $H(a)+H(b)+H(s)+2 \leq H(a)+H(b)+H(\frac{n-a-b}{3})+2$. By induction, $H(a) \leq 2 \cdot a^c - 1$, $H(b) \leq 2 \cdot b^c - 1$, and $H(\frac{n-a-b}{3}) \leq 2 \cdot (\frac{n-a-b}{3})^c - 1$, hence we need that $a^c + b^c + (\frac{n-a-b}{3})^c \leq n^c$.
We prove that $f(a,b) = a^c + b^c + (\frac{n-a-b}{3})^c$ grows monotonically with a, by assuming that $0.5 < c < 0.6$; this assumption will be verified later. We have $\frac{\partial f(a,b)}{\partial a} = c \cdot a^{c-1} - \frac{c}{3} \cdot (\frac{n-a-b}{3})^{c-1}$, which is greater than zero as long as $a^{c-1} > \frac{1}{3^c} \cdot (n-a-b)^{c-1}$. Since $c < 0.6$, we have that $1-c$ is positive, hence by raising the previous inequality to the power of $-1/(1-c)$ we get $a < 3^{c/(1-c)} \cdot (n-a-b)$, which is $a < \frac{3^{c/(1-c)}}{1+3^{c/(1-c)}} \cdot (n-b)$. By inequality (1) the latter is true as long as $\frac{n}{p} < \frac{3^{c/(1-c)}}{1+3^{c/(1-c)}} \cdot \frac{p-1}{p} \cdot n$, that is $\frac{3^{c/(1-c)}}{1+3^{c/(1-c)}} \cdot (p-1) > 1$. From $0.5 < c < 0.6$ we get that $3^{c/(1-c)} > 3$, hence $\frac{3^{c/(1-c)}}{1+3^{c/(1-c)}} > \frac{3}{4}$. Since $p > 4$, the inequality $\frac{3^{c/(1-c)}}{1+3^{c/(1-c)}} \cdot (p-1) > 1$ is satisfied, hence $\frac{\partial f(a,b)}{\partial a} > 0$ and $f(a,b)$ grows monotonically with a.
Analogously, $f(a,b)$ grows monotonically with b, as long as $0.5 < c < 0.6$.
By inequality (1) we have $a, b < n/p$, hence the monotonicity of $f(a,b)$ we proved above implies $a^c + b^c + \left(\frac{n-a-b}{3}\right)^c < 2 \cdot (n/p)^c + \left(\frac{n-2n/p}{3}\right)^c$. Thus, we need $2 \cdot (n/p)^c + \left(\frac{n-2n/p}{3}\right)^c \leq n^c$. Dividing by n^c, the inequality becomes $2 \cdot (1/p)^c + \left(\frac{1-2/p}{3}\right)^c - 1 \leq 0$. Thus, we need to choose p so to satisfy $2 \cdot (1/p)^{1/\log_2 \frac{3p}{p-1}} + \left(\frac{1-2/p}{3}\right)^{1/\log_2 \frac{3p}{p-1}} - 1 \leq 0$; the latter inequality is true[2] if $p \geq 9.956$. Thus setting $p = 9.956$ we have $H(a)+H(b)+H(s)+2 \leq 2 \cdot n^c - 1$.

From $p = 9.956$ we get $c = 0.576$. By inequality (5) the height of Γ is in $O(n^{0.576})$. This completes the proof of the height and area bounds for Γ.

Finally, we describe how to modify the construction if $x = 1$, if $x = 2$, or if x is undefined. If x is undefined, π "never turns down", that is, the construction coincides with the one above with $x = k(\pi)+1$; in this special case our construction actually coincides with the one of Frati [7]. If $x = 1$, the construction above starts from π_x, that is, it ignores the paths ρ, σ, and $(\pi_1, \ldots, \pi_{x-1})$ and their attached subtrees. If $x = 2$, π "immediately turns down": the second heaviest subtree of $r(T) = \pi_1 = \pi_{x-1}$ is drawn as above, while its lightest subtree is drawn as the second heaviest subtree of π_{x-1} above; the rest of the construction, starting from π_x, coincides with the one above. In each of these cases, the analysis on the area of the constructed drawings does not change.

3 Complete Ternary Trees

When restricting the attention to complete ternary trees better area bounds than the one from Theorem 1 are known. Namely, Frati [7] presented two inductive constructions, which are called Constructions 1 and 2 and are depicted in

[2] We used the software at www.wolframalpha.com in order to solve the inequality.

Figs. 3(a) and (b), respectively, and proved that *either* of them can be used to obtain drawings with $O(n^{1.262})$ area. That bound was improved by Ali [1], whose construction, which is depicted in Fig. 3(c), achieves an $O(n^{1.118})$ bound.

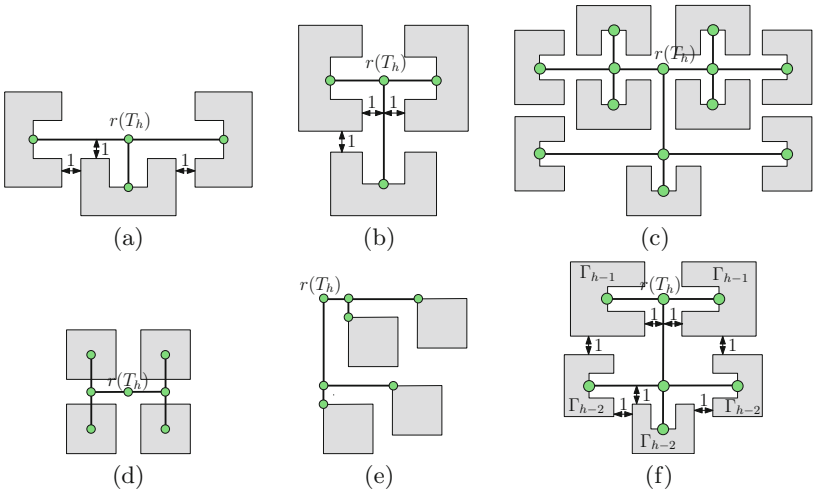


Fig. 3. (a) and (b) Constructions 1 and 2 from [7]; each of them constructs a drawing of T_h out of 3 copies of the inductively constructed drawing of T_{h-1}. (c) The construction of a drawing of T_h from [1], which employs 9 copies of the inductively constructed drawing of T_{h-2}. (d) H-drawings, as in [13]. (e) HV-drawings, as in [4]. (f) The construction of an $O(n^{1.149})$-area drawing of T_h satisfying the subtree separation property.

Denote by T_h the complete ternary tree such that every root-to-leaf path has length h. In the remainder of the paper we investigate what area bounds can be achieved for planar straight-line orthogonal drawings of T_h by combining Constructions 1 and 2. More formally, we define a *1-2 drawing* of T_h as follows. If $h = 1$, then any drawing in which the root $r(T_h)$ of T_h is at a grid point of the plane is a 1-2 drawing. If $h > 1$, then consider any three (not necessarily congruent) 1-2 drawings of T_{h-1} and arrange them as in Construction 1 or as in Construction 2; then the resulting drawing of T_h is a 1-2 drawing.

A 1-2 drawing has a feature that the drawings of Ali [1] do not have, called *subtree separation* property: the smallest axis-parallel rectangles enclosing the drawings of any two node-disjoint subtrees do not overlap. This property has been frequently considered in the tree drawing literature (see, e.g., [3,9,11,12]), because of the readability of the drawings that have it and because it is directly guaranteed by the following natural approach for drawing trees: Inductively construct drawings of the subtrees of the root and place them together so that the smallest axis-parallel rectangles enclosing them do not overlap; placing the root in the plane completes the drawing. The notorious *H-drawings* [13] and *HV-drawings* [4], which can be constructed in linear area for complete binary trees, satisfy the subtree separation property (see Figs. 3(d) and (e), respectively).

Let the *left width* of a drawing Γ of a ternary tree be the number of grid columns intersecting Γ to the left of the root; the *right width*, *top height*, and *bottom height* of Γ are defined analogously. We have the following.

Lemma 1. *Suppose that the complete ternary tree T_h has a planar straight-line orthogonal drawing Γ with the subtree separation property, with left width λ, with right width ρ, with height η, and such that, if $h \geq 2$, the three children of $r(T_h)$ are to the left, below, and to the right of $r(T_h)$.*

Then there is a 1-2 drawing Γ' of T_h with the following properties:

- *the left and right widths of Γ' are both equal to a value $\mu \leq \min\{\lambda, \rho\}$;*
- *the height of Γ' is at most η;*
- *if $h \geq 2$, then the three children of $r(T_h)$ are to the left, below, and to the right of $r(T_h)$ in Γ'; and*
- *if $h \geq 2$, then let L, B, and R be the subtrees of $r(T_h)$ rooted at the children of $r(T_h)$ to the left, below, and to the right of $r(T_h)$, respectively; then the drawings of L and R in Γ' are congruent, up to a rotation of $180°$.*

Similar statements hold true if the three children of $r(T_h)$ are below, to the right, and above $r(T_h)$, if they are to the right, above, and to the left of $r(T_h)$, or if they are above, to the left, and below $r(T_h)$.

Proof sketch: The proof proceeds in three steps. We first replace the drawings of L, B, and R in Γ by 1-2 drawings; we then replace the 1-2 drawing of one between L and R by a rotated copy of the 1-2 drawing of the other one; we finally compact the drawing of T_h by translating the drawings of L, B, and R. The full proof of the lemma will be shown in the full version of the paper.

Lemma 1 implies that 1-2 drawings require minimum area among the planar straight-line orthogonal drawings satisfying the subtree separation property.

Theorem 2. *For any positive integer h, there is a 1-2 drawing of the complete ternary tree T_h achieving minimum area among all the planar straight-line orthogonal drawings of T_h satisfying the subtree separation property.*

Proof: By Lemma 1 for any planar straight-line orthogonal drawing Γ of T_h satisfying the subtree separation property there is a 1-2 drawing Γ' whose width and height are at most equal to the width and height of Γ, respectively. □

Theorem 2 motivates the study of the area requirements of 1-2 drawings of complete ternary trees. Our next theorem proves an upper bound for the area requirements of 1-2 drawings that improves upon the $O(n^{1.262})$ area bound by Frati [7] and is close to the $O(n^{1.118})$ bound by Ali [1] for planar straight-line orthogonal drawings that do not satisfy the subtree separation property.

Theorem 3. *The n-node complete ternary tree has a 1-2 drawing (and hence a planar straight-line orthogonal drawing satisfying the subtree separation property) in $O(n^{1.149})$ area.*

Proof: We show an inductive algorithm that constructs a 1-2 drawing Γ_h of T_h in $O(n^{1.149})$ area, so that the three children $r(L)$, $r(B)$, and $r(R)$ of $r(T_h)$ are to the left, below, and to the right of $r(T_h)$, respectively. Denote by L, B, and R the subtrees of $r(T_h)$ rooted at $r(L)$, $r(B)$, and $r(R)$, respectively. Further, denote by $W(h)$ and $H(h)$ the width and the height of Γ_h, respectively.

In the base case of the algorithm we have $h \leq 2$. If $h = 1$ then Γ_1 is constructed by placing $r(T_h)$ at any grid point in the plane. If $h = 2$ then Γ_2 is constructed by combining three copies of Γ_1 by means of Construction 1. Note that $W(1) = H(1) = 1$, $W(2) = 3$, and $H(2) = 2$.

If $h > 2$ we construct Γ_h as in Fig. 3(f). In particular, we first construct a 1-2 drawing of B; this is done by combining, by means of Construction 1, three copies of the inductively constructed drawing Γ_{h-2} of T_{h-2} (two of such copies are rotated by 90°, one clockwise and one counter-clockwise). We then construct Γ_h by combining, by means of Construction 2, two copies of the inductively constructed drawing Γ_{h-1} of T_{h-1} (rotated by 90°, one clockwise and one counter-clockwise), which serve as drawings for L and R, with the just constructed drawing of B. We now analyze the area requirements of Γ_h.

The width of Γ_h is given by the maximum between the width of the part of Γ_h comprising $r(T_h)$, L, and R, and the part of Γ_h comprising B, that is:

$$W(h) = \max\{2H(h-1) + 1, W(h-2) + 2H(h-2)\}. \tag{6}$$

The height of Γ_h is given by the height of the part of Γ_h comprising $r(T_h)$, L, and R, plus the height of the part of Γ_h comprising B, that is:

$$H(h) = W(h-1) + \max\{W(h-2), (W(h-2)+1)/2 + H(h-2)\}. \tag{7}$$

We now inductively prove that $W(h) \leq k \cdot c^h - 1$ and $H(h) \leq \alpha \cdot k \cdot c^h - 1$, for any $h = 1, 2, \ldots$, where α, k, and c are suitable constants (to be determined) such that $1/2 < \alpha < 1$, $k > 1$, and $1 < c < 2$; in particular, we would like c to be as small as possible. In the base case ($1 \leq h \leq 2$), for any constants $c > 1$ and $1/2 < \alpha < 1$, a constant $k = k(c, \alpha)$ can be chosen large enough so that $W(h) \leq k \cdot c^h - 1$ and $H(h) \leq \alpha \cdot k \cdot c^h - 1$. Indeed, it suffices to choose $k \geq 4/(\alpha \cdot c)$ in order to have $k \cdot c^h - 1 > \alpha \cdot k \cdot c^h - 1 \geq 3 \geq W(1), H(1), W(2), H(2)$.

For the inductive case, assume that $W(h') \leq k \cdot c^{h'} - 1$ and $H(h') \leq \alpha \cdot k \cdot c^{h'} - 1$, for every integer $h' < h$; we prove the same inequalities for h.

By applying induction in Eq. (7), we get $H(h) \leq k \cdot c^{h-1} - 1 + \max\{k \cdot c^{h-2} - 1, (k \cdot c^{h-2} - 1 + 1)/2 + \alpha \cdot k \cdot c^{h-2} - 1\} \leq k \cdot c^{h-1} - 1 + k \cdot c^{h-2} \cdot \max\{1, \alpha + 1/2\} \leq k \cdot c^{h-1} - 1 + (\alpha + 1/2) \cdot k \cdot c^{h-2}$, where we exploited $\alpha > 1/2$. Hence, we want $k \cdot c^{h-1} - 1 + (\alpha + 1/2) \cdot k \cdot c^{h-2} \leq \alpha \cdot k \cdot c^h - 1$, that is

$$\alpha \cdot c^2 - c - (\alpha + 1/2) \geq 0. \tag{8}$$

In order to establish $W(h) \leq k \cdot c^h - 1$ we distinguish two cases.

If $W(h-2) + 2H(h-2) \geq 2H(h-1) + 1$, then by applying induction, we get $W(h) \leq k \cdot c^{h-2} - 1 + 2(\alpha \cdot k \cdot c^{h-2} - 1) = (2\alpha + 1) \cdot k \cdot c^{h-2} - 3$. Hence, we want $(2\alpha + 1) \cdot k \cdot c^{h-2} - 3 \leq k \cdot c^h - 1$, which is true as long as $c^2 \geq 2\alpha + 1$, hence

$$c \geq \sqrt{2\alpha + 1}. \tag{9}$$

If $2H(h-1)+1 > W(h-2)+2H(h-2)$, then by applying induction, we get $W(h) \leq 2(\alpha \cdot k \cdot c^{h-1}-1)+1 = 2\alpha \cdot k \cdot c^{h-1}-1$. Hence, we want $2\alpha \cdot k \cdot c^{h-1}-1 \leq k \cdot c^h - 1$, which is true as long as

$$c \geq 2\alpha. \tag{10}$$

Now pick $c = 2\alpha$, thus satisfying inequality (10). Substituting $c = 2\alpha$ in inequality (8), we want $4\alpha^3 - 3\alpha - 1/2 \geq 0$, which is true if $\alpha \geq 0.9397$. So take $\alpha = 0.9397$, implying that inequality (8) is satisfied and note that $c = 2\alpha = 1.8794 > 1.6969 > \sqrt{2\alpha+1}$, hence inequality (9) is satisfied as well. This completes the induction and hence proves that $W(h), H(h) \in O(1.8794^h)$.

Since $h = \log_3 n + O(1)$, we have $W(h), H(h) \in O(1.8794^{\log_{1.8794} n / \log_{1.8794} 3})$, hence $W(h), H(h) \in O(n^{0.5744})$. Theorem 3 follows. □

We now prove the following theorem.

Theorem 4. *A minimum-area 1-2 drawing of a complete ternary tree can be computed in polynomial time.*

The proof of Theorem 4 is based on the following strategy[3], which resembles the approach proposed in [8] in order to compute minimum-area *LR-drawings* of binary trees. For any value of h we aim at computing all the *Pareto-optimal* width-height pairs (ω, η) for the 1-2 drawings of T_h; these are the pairs such that: (i) T_h admits a 1-2 drawing with width ω and height η; and (ii) there exists no pair (ω', η') such that T_h admits a 1-2 drawing with width ω' and height η', where $\omega' \leq \omega$, $\eta' \leq \eta$, and at least one of these inequalities is strict.

Lemma 2. *There are $O(n)$ Pareto-optimal width-height pairs for the 1-2 drawings of a complete ternary tree with n nodes.*

Proof: The statement comes from the following two observations. First, for any integer value ω there is at most one Pareto-optimal pair (ω, η). Second, any Pareto-optimal pair (ω, η) has $\omega \leq n$, as any 1-2 drawing with width greater than n has a grid column not containing any vertex; then the part of the drawing to the right of such a grid column can be moved one unit to the left, resulting in a 1-2 drawing with the same height and with smaller width. □

Lemma 3. *The Pareto-optimal width-height pairs for the 1-2 drawings of T_h can be computed in polynomial (in the number of nodes of T_h) time.*

Proof: In this proof by *pair* we always mean Pareto-optimal width-height pair. Suppose that the pairs for the 1-2 drawings of T_{h-1} have been computed already (note that $(1,1)$ is the only pair for the 1-2 drawings of T_1). We compute the pairs for the 1-2 drawings of T_h by considering all the triples (p_l, p_b, c) where

[3] We claim that Theorem 4 can be generalized to ternary trees that are not necessarily complete. However, since our main interest in 1-2 drawings comes from the study of the area requirements of complete ternary trees, we opted for keeping the exposition simple and present the theorem and its proof for complete ternary trees only.

$p_l = (\omega_l, \eta_l)$ and $p_b = (\omega_b, \eta_b)$ are pairs for the 1-2 drawings of T_{h-1} and c is either 1 or 2. By Lemma 2 there are $O(n^2)$ such triples. For each triple (p_l, p_b, c) we consider the 1-2 drawing Γ defined as follows:

- the drawing of L in Γ is a 1-2 drawing with width ω_l and height η_l;
- the drawing of R in Γ is a 1-2 drawing with width ω_l and height η_l;
- the drawing of B in Γ is a 1-2 drawing with width ω_b and height η_b;
- the drawings of L, B, and R are arranged as in Construction c (the drawings of L and R are rotated clockwise and counter-clockwise by 90°, respectively).

h	1	2	3	4	5	6	7	8	9	10	11	12	13	14
n	1	4	13	40	121	364	1093	3280	9841	29 524	88 573	265 720	797 161	2 391 484
Area	1	6	25	99	342	1184	4030	13 320	44 457	144 690	469 221	1 520 189	4 840 478	15 550 542

h	15	16	17	18	19	20
n	7 174 453	21 523 360	64 570 081	193 710 244	581 130 733	1 743 392 200
Area	49 461 933	157 388 427	498 895 215	1 580 110 511	4 990 796 080	15 765 654 805

The width and height of Γ can be computed easily as follows (see also [7]). If $c = 1$, then the width of Γ is equal to $2\eta_l + \omega_b$, otherwise it is equal to the maximum between $2\eta_l + 1$ and ω_b. Further, if $c = 1$, then the height of Γ is equal to the maximum between ω_l and $\eta_b + (\omega_l + 1)/2$, otherwise it is equal to $\omega_l + \eta_b$.

Out of the $O(n^2)$ 1-2 drawings of T_h constructed as above we only keep the $O(n)$ drawings corresponding to Pareto-optimal width-height pairs – it comes again from Lemma 2 that there are this many Pareto-optimal width-height pairs. This can be accomplished in polynomial time by ordering the $O(n^2)$ width-height pairs by increasing width and, secondarily, by increasing height, and by removing every width-height pair that is preceded by a pair with smaller or equal height.

The proof of correctness of the described algorithm exploits Lemma 1. In particular, the algorithm uses, in every constructed drawing of T_h, two drawings for the left and right subtrees of $r(T_h)$ that are congruent, up to a rotation of 180°, which can be done without loss of generality by Lemma 1. Further, the algorithm constructs the Pareto-optimal pairs for the 1-2 drawings of T_h starting from the Pareto-optimal pairs for the 1-2 drawings of T_{h-1}. This is also not a loss of generality since, again by Lemma 1, any 1-2 drawing of T_{h-1} can be replaced by a 1-2 drawing of T_{h-1} corresponding to a Pareto-optimal pair in which the bottom and top heights (or the left and right widths) are the same, without increasing the width and height of the drawing of T_h. □

Lemmata 2 and 3 imply Theorem 4, as the minimum area for a 1-2 drawing of T_h is equal to $\min\{\omega \cdot \eta\}$, where the minimum is taken over all the Pareto-optimal width-height pairs (ω, η) for the 1-2 drawings of T_h.

We run a mono-thread C implementation of the algorithm that computes the Pareto-optimal width-height pairs for the 1-2 drawings of T_h described in

the proof of Lemma 3 on a machine with two 4-core 3.16 GHz Intel(R) Xeon(R) CPU X5460 processors, with 48 GB of RAM, running Ubuntu 14.04.2 LTS. We could compute the Pareto-optimal width-height pairs (ω, η) and the minimum area for the 1-2 drawings of T_h with h up to 20. The computation of the pairs for $h = 20$ took roughly 5 days. The table below shows the value of h, the corresponding value of n, which is $(3^h - 1)/2$, and the minimum area required by any 1-2 drawing (and by Theorem 2 by any planar straight-line orthogonal drawing satisfying the subtree separation property) of T_h.

By means of the Mathematica software [16] we searched for the function of the form $a \cdot n^b + c$ that better fits the values of the table above, according to the least squares optimization method. The optimal function we got is $3.3262 \cdot x^{1.047} - 181\,209.1337$. While the large absolute value of the additive constant suggests the need for a lower order term or for a different optimization method, the experimentation also seems to indicate that planar straight-line drawings with the subtree separation property cannot be constructed within almost-linear area. We hence conclude the paper with the following conjecture.

Conjecture 1. There exists a constant $\varepsilon > 0$ such that n-node complete ternary trees require $\Omega(n^{1+\epsilon})$ area in any planar straight-line orthogonal drawing satisfying the subtree separation property.

References

1. Ali, A.: Straight line orthogonal drawings of complete ternery trees. MIT Summer Program in Undergraduate Research Final Paper, July 2015
2. Chan, T.M.: Tree drawings revisited. In: Speckmann, B., Tóth, C.D. (eds.) Symposium on Computational Geometry (SoCG 2018) (2018)
3. Chan, T.M., Goodrich, M.T., Kosaraju, S.R., Tamassia, R.: Optimizing area and aspect ratio in straight-line orthogonal tree drawings. Comput. Geom. **23**(2), 153–162 (2002)
4. Crescenzi, P., Di Battista, G., Piperno, A.: A note on optimal area algorithms for upward drawings of binary trees. Comput. Geom. **2**, 187–200 (1992)
5. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice-Hall, Englewood (1999)
6. Di Battista, G., Frati, F.: Drawing trees, outerplanar graphs, series-parallel graphs, and planar graphs in a small area. In: Pach, J. (ed.) Thirty Essays on Geometric Graph Theory, pp. 121–165. Springer, New York (2013). https://doi.org/10.1007/978-1-4614-0110-0_9
7. Frati, F.: Straight-line orthogonal drawings of binary and ternary trees. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 76–87. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77537-9_11
8. Frati, F., Patrignani, M., Roselli, V.: LR-drawings of ordered rooted binary trees and near-linear area drawings of outerplanar graphs. In: Klein, P.N. (ed.) Symposium on Discrete Algorithms (SODA 2017), pp. 1980–1999. SIAM (2017)
9. Garg, A., Rusu, A.: Straight-line drawings of binary trees with linear area and arbitrary aspect ratio. In: Goodrich, M.T., Kobourov, S.G. (eds.) GD 2002. LNCS, vol. 2528, pp. 320–332. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36151-0_30

10. Nomura, K., Tayu, S., Ueno, S.: On the orthogonal drawing of outerplanar graphs. IEICE Trans. 88-A(6), 1583–1588 (2005)
11. Rusu, A.: Tree drawing algorithms. In: Tamassia, R. (ed.) Handbook of Graph Drawing and Visualization, pp. 155–192. CRC Press (2016). Chap. 5
12. Rusu, A., Santiago, C.: Grid drawings of binary trees: an experimental study. J. Graph Algorithms Appl. **12**(2), 131–195 (2008)
13. Shiloach, Y.: Linear and Planar Arrangement of Graphs. Ph.D. thesis, Weizmann Institute of Science, Rehovot (1976)
14. Shin, C., Kim, S.K., Chwa, K.: Area-efficient algorithms for straight-line tree drawings. Comput. Geom. **15**(4), 175–202 (2000)
15. Tamassia, R.: On embedding a graph in the grid with the minimum number of bends. SIAM J. Comput. **16**(3), 421–444 (1987)
16. Wolfram Research Inc.: Mathematica 10 (2014). http://www.wolfram.com

A Fixed-Parameter Algorithm for the Max-Cut Problem on Embedded 1-Planar Graphs

Christine Dahn(✉), Nils M. Kriege, and Petra Mutzel

Department of Computer Science, TU Dortmund University, Dortmund, Germany
{christine.dahn,nils.kriege,petra.mutzel}@cs.tu-dortmund.de

Abstract. We propose a fixed-parameter tractable algorithm for the Max-Cut problem on embedded 1-planar graphs parameterized by the crossing number k of the given embedding. A graph is called 1-planar if it can be drawn in the plane with at most one crossing per edge. Our algorithm recursively reduces a 1-planar graph to at most 3^k planar graphs, using edge removal and node contraction. The Max-Cut problem is then solved on the planar graphs using established polynomial-time algorithms. We show that a maximum cut in the given 1-planar graph can be derived from the solutions for the planar graphs. Our algorithm computes a maximum cut in an embedded 1-planar graph with n nodes and k edge crossings in time $\mathcal{O}(3^k \cdot n^{3/2} \log n)$.

Keywords: Maximum cut · Fixed-parameter tractable
1-planar graphs

1 Introduction

Partitioning problems on graphs receive increasing attention in the literature. Here the task is to partition the set of nodes of a given (weighted) undirected graph so that the number (or weighted sum) of connections between the parts is minimized. A special case is the Max-Cut problem which asks for a node partition into two sets so that the sum of the edge weights in the cut is maximised. The problem is getting increasing attention in the literature, since it is directly related to solving Ising spin glass models (see, e.g., Barahona [3]) which are of high interest in physics. Besides its theoretical merits, Ising spin glass models need to be solved in adiabatic quantum computation [20]. Other applications occur in the layout of electronic circuits [6,8].

The Max-Cut problem has been shown to be NP-hard for general graphs [15]. Moreover, Papadimitriou and Yannakakis [24] have shown that the Max-Cut problem is APX-hard, i.e., there does not exist a polynomial-time approximation scheme unless P = NP. Goemans and Williamson suggested a randomized constant factor approximation algorithm [10] which has been derandomized by Mahajan and Ramesh [19] and has a performance guarantee of 0.87856.

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 141–152, 2018.
https://doi.org/10.1007/978-3-319-94667-2_12

There are a number of special cases for which the problem can be solved in polynomial time. The most prominent case arises if the weights of all edges are negative, since then the problem can be solved via network flow. Other special cases are graphs without long odd cycles [11] or weakly bipartite graphs [12]. Another prominent case arises for planar input graphs. Orlova and Dorfman [22] and Hadlock [13] have shown that the Max-Cut problem can be solved in polynomial time for unweighted planar graphs. Their algorithms can be extended to work on weighted planar graphs (e.g., Mutzel [21]). Currently, the fastest algorithms have been suggested by Shih et al. [25] and by Liers and Pardella [18]. These results have been extended to the class of graphs not contractible to K_5 [5] and to toroidal graphs [4,9] (i.e., graphs that can be embedded on the torus). In this paper, we show an extension of the planar special case to that of the class of 1-planar graphs.

A graph is 1-planar if it can be drawn into the plane so that every edge is crossed at most once. While planarity testing can be done in linear time [14], the recognition problem for 1-planar graphs is much harder. Korzhik and Mohar showed that 1-planarity testing is NP-hard [16]. However, there are fixed-parameter tractable (FPT) algorithms parameterized by the cyclomatic number (the minimum number of edges that must be removed from the graph to create a forest), the tree-depth or the node cover number [2]. For 1-planar graphs these algorithms construct a 1-planar embedding.

Our Contribution. Given an embedded 1-planar graph with k crossings, we suggest a fixed-parameter tractable algorithm for the Max-Cut problem with parameter k. The idea of our algorithm is to recursively reduce the input graph into a set of at most 3^k planar graphs using a series of edge removals and node contractions. The planar instances can then be solved using the polynomial time algorithms suggested in [18,25] with running time $O(n^{3/2} \log n)$ for a planar graph with n nodes.

The paper is organized as follows. Section 2 contains the basic definitions concerning cuts and 1-planarity. We also introduce the class of k-almost-planar graphs which have 1-planar drawings not exceeding k crossings. In Sect. 3 we present our new algorithm for embedded 1-planar graphs and prove its correctness. Our analysis of its running time shows that it is fixed-parameter tractable with parameter k. We end with a conclusion and open problems in Sect. 4.

2 Preliminaries

Throughout our paper, we consider undirected weighted graphs $G = (V, E, c)$ with arbitrary edge weights. A partition of the nodes of G into two sets $S \subseteq V$ and $\overline{S} = V \backslash S$ defines the *cut* $\delta(S, G) = \{(uv) \in E \mid (u \in S \text{ and } v \in \overline{S}) \text{ or } (v \in S \text{ and } u \in \overline{S})\}$. The *value of a cut* $\delta(S, G)$ in the graph G is the sum of weights of all edges in the cut: $c(\delta(S, G)) = \sum_{e \in \delta(S,G)} c_e$. The Max-Cut problem searches for a cut in a given weighted graph with highest value. For the graph class of planar graphs, the Max-Cut problem can be solved in polynomial time.

A graph is *planar* if it admits a *planar drawing*, i.e., a drawing on the plane without any edge crossing. A drawing admits a *rotation system* which is a clockwise-ordering of the incident edges for every node. In a planar drawing, a rotation system defines the *faces*, i.e., the topologically connected regions of the plane. One of the faces, the *outer face*, is unbounded. A face is uniquely described by its boundary edges. Such a description for each face is an equivalent definition of a (planar) embedding. A *(planar) embedding* represents the set of all planar drawings with the same faces. It can be represented by the description of the faces or by the rotation system. It is well known that planarity testing can be solved in linear time [14]. The same is true for computing a planar embedding. In order to generate crossing free drawings of planar graphs, a number of various algorithms exist, e.g., the straight-line drawing algorithm by de Fraysseix et al. [7].

Planar graphs are contained in the class of 1-planar graphs. A graph is *1-planar*, if it admits a *1-planar drawing*, i.e., a drawing on the plane with at most one crossing per edge. Testing 1-planarity is NP-hard [16] even in the case of bounded treewidth or bandwith [2]. A *1-planar embedding* defines the faces of a given 1-planar drawing and can be represented by the set of crossings X and a list of edges and edge segments (half edges) for the crossings for each face. Note that a 1-planar embedding uniquely defines a rotation system for the nodes. However, the opposite is not true. In general, a rotation system does not allow for computing the crossings efficiently or a 1-planar embedding. Auer et al. [1] have shown that testing 1-planarity of a graph with a fixed rotation system is NP-hard even if the graph is 3-connected.

We call a 1-planar graph *k-almost-planar* if it admits a 1-planar drawing with at most k edge crossings. For edge removal and node contraction we use the following notation: $G - e = (V, E \setminus \{e\})$ denotes the graph obtained from $G = (V, E)$ by deleting the edge $e \in E$. G/xy contracts the two nodes x and y into a new node $v_{xy} \notin V$. In doing so, the edges leading to x or y are replaced by a new edge to v_{xy}. Multi-edges to v_{xy} are contracted to one edge and their edge weights are added, self-loops are deleted. We denote the inverse operation of contraction by Split. The contraction and Split operation can be applied to a subset of nodes $S \subseteq V$:

$$S/xy = \begin{cases} S \setminus \{x, y\} \cup \{v_{xy}\} & \text{if } x, y \in S \\ S & \text{otherwise} \end{cases}$$

$$\textsc{Split}(S, v_{xy}) = \begin{cases} S \setminus \{v_{xy}\} \cup \{x, y\} & \text{if } v_{xy} \in S \\ S & \text{otherwise} \end{cases}$$

3 Max-Cut for Embedded 1-Planar Graphs

Our main idea for computing the maximum cut in an embedded 1-planar graph G is to eliminate its k crossings and then use a Max-Cut algorithm for planar graphs on the resulting planar graph. In order to remove a crossing, we need to

know the two crossing edges of each crossing. We use two methods to remove a crossing: Either by deleting one of the crossing edges, or by contracting two nodes that do not belong to the same crossing edge.

3.1 Removing the Crossings

In this section let $G = (V, E, c)$ be a k-almost-planar graph with a 1-planar embedding (Π, X) and a set of crossing edges X with $|X| = k$. A crossing is defined by a pair of crossing edges, e.g., let $\chi = \{e_{vy}, e_{wz}\} \in X$ be an arbitrary crossing. The following lemma shows that specific node contractions (and edge deletions) remove at least one crossing and do not introduce new crossings. Figure 1b and c show examples of node contraction and Fig. 1d shows an example of edge deletion.

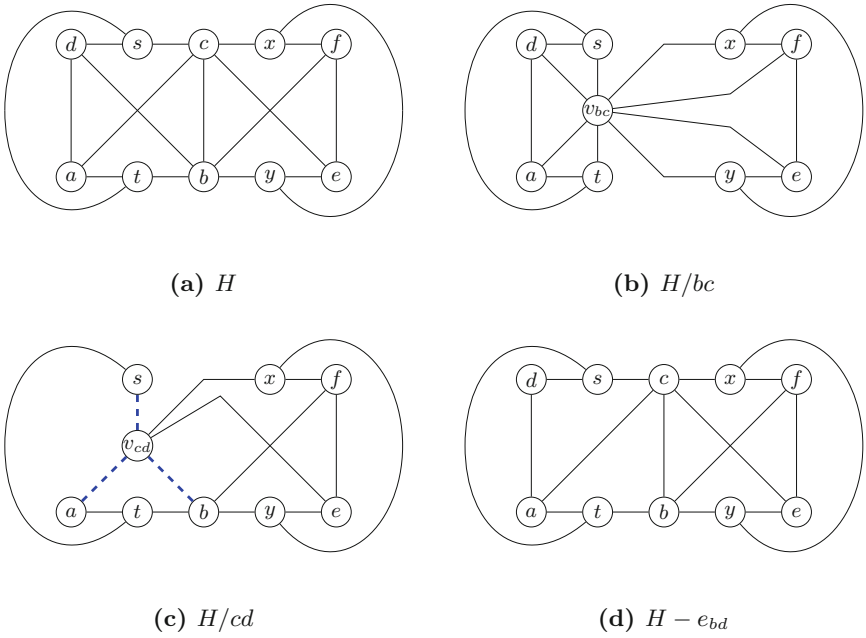
Lemma 1. *Let G be a k-almost-planar graph with 1-planar embedding (Π, X) and $\chi = \{e_{vy}, e_{wz}\} \in X$ be an arbitrary crossing. The graphs G/ab, $G - e_{vy}$ and $G - e_{wz}$ are $(k-1)$-almost-planar for $ab \in \{vw, vz, wy, yz\}$. The set of crossings in the resulting 1-planar embedding is a proper subset of X.*

Proof. Since the contracted nodes a and b are each an endpoint to one of the crossing edges, the contracted node is an endpoint to both edges. Since e_{vy} and e_{wz} now have a common endpoint, they can be drawn without a crossing. Therefore the crossing χ is removed. The contraction does not create new crossings because the two nodes a and b can be moved along their half edges towards the crossing. This is possible because we have a 1-planar embedding which has the property that every crossing is incident to two half edges connecting it with its endpoints. The new node v_{ab} is then placed where the crossing used to be. All other edges can be extended to the new node along the way of the same half edges without creating new crossings. Multi-edges are merged into a single edge and self-loops are deleted. In $G - e_{vy}$ and $G - e_{wz}$, the crossing χ is removed by deleting one of its crossing edges. Obviously this does not lead to new crossings. So in both cases the number of crossings decreases. □

The recursive application of Lemma 1 shows that all crossings can be removed with these two operations. Thus after k contraction or removal operations, the resulting graph is planar and a planar Max-Cut algorithm can be applied to compute a maximum cut. The following lemma shows how to project a cut in G/xy or $G - e$ back onto G.

Lemma 2. *Let $G = (V, E, c)$ be a weighted graph, $x \neq y \in V$ and $S \subseteq V$.*

(i) Let $\delta(S, G/xy)$ be a cut in G/xy, then the cut $\delta(\textsc{Split}(S, v_{xy}), G)$ in G has the same value.
(ii) If x and y are in the same set ($x, y \in S$ or $x, y \in \overline{S}$) then $\delta(S, G) = \delta(S, G - e_{xy})$ for $e_{xy} \in E$.



(a) H **(b)** H/bc

(c) H/cd **(d)** $H - e_{bd}$

Fig. 1. An example how a crossing can be removed. (Blue dashed edges are merged edges from H.) (Color figure online)

Proof. (i) Let S define a cut in G/xy. If the contracted node v_{xy} is split, the cut is projected from G/xy to G. The corresponding set of nodes in G is $S' = \textsc{Split}(S, v_{xy})$. It defines a cut in G. If $v_{xy} \notin S$ then $S = S'$. The weight of an edge $e \in \delta(S, G/xy)$ in G/xy is either the same as the weight of the corresponding edge $e' \in \delta(S', G)$ in G or it is split between two edges $e', e'' \in \delta(S', G)$ in G. The only edge that might exist in G but not in G/xy is e_{xy}. Since x and y were contracted in G/xy, they are either both in S' or both in $\overline{S'}$. Therefore the only edge that could be added in G by splitting v_{xy} can not add to the value of $\delta(S', G)$ in G. So no weights are lost or added due to the projection and the two cuts have the same value.
(ii) This is obvious because e_{xy} is in neither of the two cuts. □

3.2 The Max-Cut Algorithm

We use the three operations introduced above to successively remove all crossings of a 1-planar graph. All planar instances obtained in this way are then solved by a Max-Cut algorithm for planar graphs. From the solutions of the planar graphs, we construct a solution for the original graph. Note that the algorithm only needs the graph G and the set of edge crossings X as input. However, the 1-planar embedding is needed to show the correctness of the algorithm.

`MaxCut`(G, X)

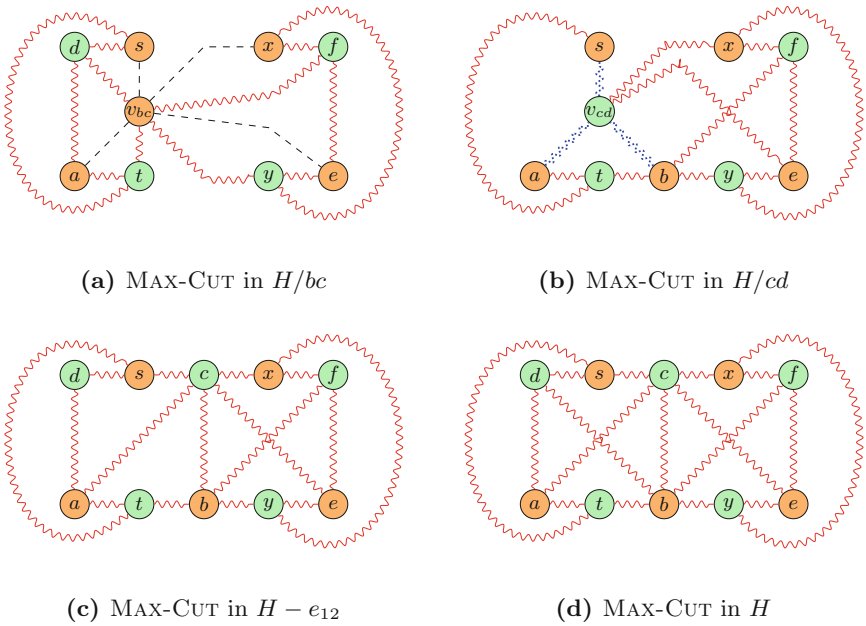
Input: An undirected weighted 1-planar graph G and a set of crossing edges X in a 1-planar embedding of G.
Output: A set $S \subseteq V_G$ defining a maximum cut $\delta(S, G) \subseteq E_G$ in G.

```
1: if X = ∅ then
2:     S ← MaxCut_planar(G)
3: else
4:     choose an element χ ← {e_vy, e_wz} ∈ X
5:     S_1 ← MaxCut(G/wy, Update(X, w, y))
6:     S_2 ← MaxCut(G/yz, Update(X, y, z))
7:     S_3 ← MaxCut(G − e_wz, X \ {χ})
8:     G_1 ← G/wy, G_2 ← G/yz, G_3 ← G − e_wz
9:     j ← argmax_{1≤i≤3} c(δ(S_i, G_i))
10:    if j = 1 then
11:        S ← Split(S_1, v_wy)
12:    else if j = 2 then
13:        S ← Split(S_2, v_yz)
14:    else
15:        S ← S_3
16:    end if
17: end if
18: return S
```

Algorithm 1.1. Max-Cut algorithm for embedded 1-planar graphs

Algorithm 1.1 realizes this approach with a recursive function, which is initially called with the input graph G and the set of crossings X present in its embedding. As the algorithm progresses, the graph is successively modified and the set of crossings is adjusted according to the modifications applied. If the graph G passed as parameter to the function is planar ($X = \emptyset$), then a planar Max-Cut algorithm is called (line 2). If there are still crossings remaining, an arbitrary crossing is selected and removed in three different ways: Let y be an arbitrary endpoint of one crossing edge and e_{wz}, $w \neq y$, $z \neq y$, the other crossing edge, then (i) the nodes y and w are contracted, (ii) the nodes y and z are contracted, and (iii) the edge e_{wz} is deleted. Each operation removes at least the selected crossing, but in case (i) and (ii) also other crossing may be affected. Therefore, the set of crossings X is adjusted by the function `Update`. If two nodes w, y are contracted, `Update`(X, w, y) removes every crossing in X which was dissolved by contracting w and y, and replaces every appearance of w or y in X with the contracted node v_{wy}. To check if a crossing was dissolved, `Update` checks if w and y are both part of the crossing. Since every crossing needs to be checked once, `Update` has a linear running time. For each case, the recursive function is called with the modified graph and the set of crossings as a parameter (lines 5–7). Each call returns a node set defining a maximum cut in the modified instance. The cut with maximal value is then projected back to G. If the maximum cut is obtained in a graph with contracted nodes, i.e., case (i)

or (ii), then the original nodes are restored by the function $\textsc{Split}(S, v_{wy})$, which replaces v_{wy} with w and y if S contains the contracted node. This cut-defining set is then returned as the solution to the subproblem.



(a) Max-Cut in H/bc **(b)** Max-Cut in H/cd

(c) Max-Cut in $H - e_{12}$ **(d)** Max-Cut in H

Fig. 2. An example how the algorithm calculates a Max-Cut in an embedded 2-almost-planar graph. (Blue dotted edges were merged and have weight 2; all other edges have weight 1; curvy edges belong to the cut; black dashed edges do not belong to the cut.) (Color figure online)

Example 1. Given the 2-almost planar graph H in Fig. 1a with uniform edge weights, e.g. 1. The algorithm removes the left crossing in the three described ways. The resulting graphs are shown in Fig. 1b–d. The recursively calculated cuts of these graphs are depicted in Fig. 2a–c with 2b being the largest cut. This cut is transferred back to H by splitting the contracted node v_{cd}. The resulting cut is shown in Fig. 2d. It is a maximum cut in H.

3.3 Correctness

The four endpoints of a crossing can be partitioned in eight non-isomorphic ways, cf. Fig. 3: (a) all endpoints in one set, (b)/(c)/(d)/(e) three endpoints in one set without $v/w/y/z$, (f)/(g) the two endpoints of different crossing edges in the same sets, or (h) the two endpoints of the same crossing edges in one set each. For arbitrary graphs, the induced cut is different because non-crossing edges might be replaced with a path or might not exist at all.

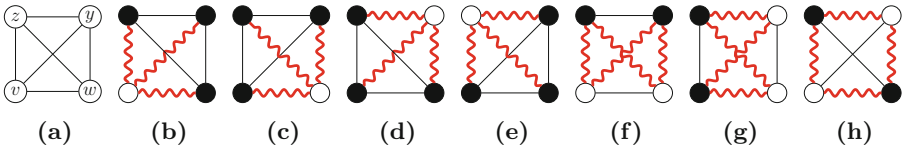


Fig. 3. The 8 non-isomorphic partitions of the four endpoints of a crossing. (The red and curvy edges belong to the cut that is defined by the corresponding partition on the K_4.) (Color figure online)

Lemma 3. *Let $G = (V, E, c)$ be a 1-planar graph with a 1-planar embedding (Π, X), $S \subseteq V$, and $\chi = \{e_{vy}, e_{wz}\} \in X$ be an arbitrary crossing.*

(i) If a cut $\delta(S, G)$ in G separates the four endpoints of χ as shown in Fig. 3(a), (b), (c) or (f) then $S_2 = S/yz$ defines a cut in G/yz with the same value. If $\delta(S, G)$ is maximal in G so is $\delta(S_2, G/yz)$ in G/yz.

(ii) If a cut $\delta(S, G)$ in G separates the four endpoints of χ as shown in Fig. 3(a), (b), (e) or (g) then $S_1 = S/wy$ defines a cut in G/wy with the same value. If $\delta(S, G)$ is maximal in G so is $\delta(S_1, G/wy)$ in G/wy.

(iii) If a cut $\delta(S, G)$ in G separates the four endpoints of χ as shown in Fig. 3(a), (b), (d) or (h) then $S_3 = S$ defines a cut in $G - e_{wz}$ with the same value. If $\delta(S, G)$ is maximal in G, so is $\delta(S_3, G - e_{wz})$ in $G - e_{wz}$.

Proof. (i) Let S define a cut in G that separates the endpoints of χ as shown in Fig. 3(a), (b), (c) or (f). By contracting y and z, the set of nodes is projected to G/yz and $\delta(S_2, G/yz)$ is a cut in G/yz. The only edge that might have been removed in G/yz does not add to the value of $\delta(S, G)$ in G because y and z are not separated by the cut (Fig. 3(a), (b), (c) or (f)). Therefore, the two cuts have the same value in both graphs. Let S define a maximum cut in G (with the required property). If there was a cut $\delta(S', G/yz)$ in G/yz larger than $\delta(S_2, G/yz)$, then $\textsc{Split}(S', v_{yz})$ would define a cut in G with the same value as $\delta(S', G)$ (Lemma 2(i)), contradicting that $\delta(S, G)$ is maximal in G.
(ii) The proof of the second proposition is analogous to the proof of the first.
(iii) Let S define a cut in G that separates the endpoints of χ as shown in Fig. 3(a), (b), (d) or (h). Since G and $G - e_{wz}$ have the same set of nodes, $\delta(S_3, G - e_{wz})$ is a cut in $G - e_{wz}$ as well. We know that w and z are not separated by the cut (Fig. 3(a), (b), (d) or (h)). Therefore the only edge that was removed in $G - e_{wz}$ does not add to the value of the cut in G and the cut has the same value in both graphs. Let S define a maximum cut in G (with the required property). If there was a cut $\delta(S', G - e_{wz})$ in $G - e_{wz}$ larger than $\delta(S_3, G - e_{wz})$, then $\delta(S', G)$ would be a cut in G as well (Lemma 2(ii)), contradicting that $\delta(S, G)$ is maximal in G. □

Theorem 1. *Algorithm 1.1 computes a maximum cut in a 1-planar graph G, given a set of crossing edges X in a 1-planar embedding of G.*

Proof. We prove its optimality by induction over k. For $k = 0$, the given graph is planar. Thus the Max-Cut algorithm for planar graphs calculates a node set

defining a maximum cut in G. Let S^* define a maximum cut in G. We show that the cut $\delta(S, G)$ defined by the calculated node set S is not smaller than $\delta(S^*, G)$. Let $G_1 = G/wy, G_2 = G/yz$ and $G_3 = G - e_{wz}$ be the $(k-1)$-almost-planar graphs (Lemma 1) whose cuts $\delta(S_1, G_1), \delta(S_2, G_2)$ and $\delta(S_3, G_3)$ are calculated recursively by the algorithm. There are 8 possible ways for S^* to separate the four endpoints of χ. These are shown in Fig. 3(a)–(h). If the endpoints of χ are separated as shown in (a), (b), (e) or (g), then $\delta(S^*, G)$ has the same value as a maximum cut $\delta(S_1^*, G_1)$ in G_1 (Lemma 3(ii)). Due to the induction hypothesis, $\delta(S_1, G_1)$ is not smaller than $\delta(S_1^*, G_1)$. If the endpoints of χ are separated as shown in (c) or (f), then $\delta(S^*, G)$ has the same value as a maximum cut $\delta(S_2^*, G_2)$ in G_2 (Lemma 3(i)). Due to the induction hypothesis, $\delta(S_2, G_2)$ is not smaller than $\delta(S_2^*, G_2)$. If the endpoints of χ are separated as shown in (d) or (h), then $\delta(S^*, G)$ has the same value as a maximum cut $\delta(S_3^*, G_3)$ in G_3 (Lemma 3(iii)). Due to the induction hypothesis, $\delta(S_3, G_3)$ is not smaller than $\delta(S_3^*, G_3)$. The algorithm chooses the node set defining the largest of these three cuts (line 9–16) and projects it back to G without changing its value (Lemma 2). Thus the calculated cut $\delta(S, G)$ is not smaller than $\delta(S_1, G_1), \delta(S_2, G_2)$ and $\delta(S_3, G_3)$. □

3.4 Running Time

Let n be the number of nodes and m be the number of edges of a given graph. It is well known that a 1-planar graph has at most $4n - 8$ edges [23]. For an arbitrary 1-planar drawing, the number of crossings is bounded by $\frac{m}{2}$, since every edge can be crossed at most once and every crossing needs two edges. With the previous observation, we can establish a bound depending on the number of nodes: $k \leq 2n - 4$.

Theorem 2. *Algorithm 1.1 computes a maximum cut in an embedded 1-planar graph with n nodes and k crossings in time $\mathcal{O}(3^k \cdot (T_p(n)+n))$, where $T_p(n)$ is the running time of a planar* Max-Cut *algorithm. Using the algorithms suggested in [17] or [25], the running time is $\mathcal{O}(3^k \cdot n^{3/2} \log n)$.*

Proof. Let $T(k, n)$ be the running time of Algorithm 1.1 on an embedded 1-planar graph G with n nodes and k crossings. If G is planar, our algorithm uses a planar Max-Cut algorithm, resulting in $T(0, n) = T_p(n)$. `Update` has a linear running time of $\mathcal{O}(k)$, since every crossing in X needs to be checked only once. The contractions of G/wy and G/yz take time $\mathcal{O}(n + m)$ and the edge removal $G - e_{wz}$ takes time $\mathcal{O}(m)$. Reversing a contraction on a set of nodes S_i with `Split` takes $|S_i|$ steps, resulting in a running time of $\mathcal{O}(n)$. Hence the recursive running time is:

$$T(k, n) = 3 \cdot T(k-1, n) + \mathcal{O}(k + n + m)$$

An induction proof shows that $T(k, n) = 3^k \cdot (T(0, n) + \sum_{i=1}^{k} 3^{-i} \cdot \mathcal{O}(i + n + m))$. Since m is bounded by $4n - 8$ [23], k is bounded by $2n - 4$ (see above), i is bounded by k and the geometric sum equals a value between 0 and 1, the overall running time is $\mathcal{O}(3^k \cdot (T_p(n) + n))$. Liers and Pardella [17] or Shih et al. [25]

describe a planar Max-Cut algorithm with a running time of $\mathcal{O}(n^{3/2} \cdot \log n)$, resulting in a concrete running time of $\mathcal{O}(3^k \cdot n^{3/2} \log n)$ for our algorithm. □

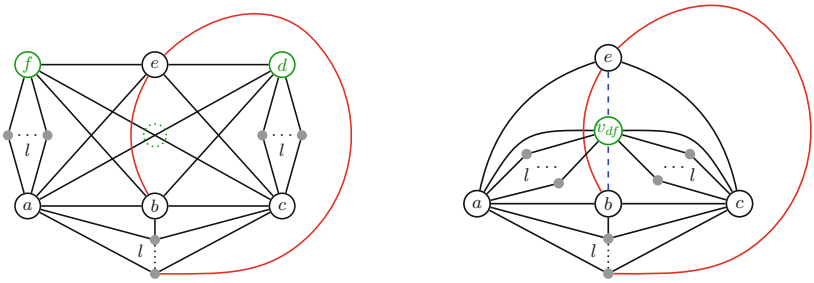
If the number of crossings k in a 1-planar embedding is fixed, the running time of Algorithm 1.1 is polynomial. However, in an arbitrary 1-planar embedding, k is not fixed and the factor 3^k leads to an exponential worst case running time. But we can show that our algorithm is *fixed-parameter tractable* with parameter k. Since its running time can be split into an exponential part, depending only on the parameter k, (3^k) and a polynomial part in the size of the input graph $(T_p(n) + n)$, the algorithm is *fixed-parameter tractable* with parameter k.

Theorem 3. *The* Max-Cut *problem on embedded 1-planar graphs is fixed-parameter tractable parameterized by the crossing number k of the given 1-planar embedding.*

4 Conclusion and Open Problems

We have presented a polynomial time algorithm for computing a Max-Cut in a 1-planar graph provided with a 1-planar embedding with a constant number of crossings. This shows that the Max-Cut problem on embedded 1-planar graphs is in the class FPT.

The question arises if our approach can be extended to general graphs with up to k crossings per edge, so called *k-planar graphs*. Our approach is based on the fact that node contractions and edge deletions decrease the number of crossings (see Lemma 1). Figure 4 shows that this is no longer true if an edge is crossed more than once. In this case, there are crossings that do not have direct half edges connecting it to its endpoints like, e.g., the crossing (ad, cf) in Fig. 4.



(a) G **(b)** G/df

Fig. 4. A 4-planar graph where the contraction of the nodes d and f leads to $\mathcal{O}(l)$ new crossings. The two edges that generate the new crossings are drawn in red. Between a and f (resp. c and d) in G are l independent paths. Beneath b there are l paths between a and c that are pairwise connected and therefore have a specific order. The highest path contains a node connected to b and the lowest path contains a node connected to e. No matter where e is drawn in G/df, one of the two red edges crosses at least $l - 1$ other edges. (Color figure online)

If we contract d and f, we get plenty of new crossings in the new graph G/df. We are currently working to generalize our approach to embedded k-planar graphs.

Another interesting question would be to drop the assumption that we are given a 1-planar embedding. Note that our algorithm does not need such an embedding as input, it only needs to get a list of edge crossings that must correspond to a 1-planar embedding. However, for our correctness analysis it is important to have a 1-planar embedding of the graph.

References

1. Auer, C., Brandenburg, F.J., Gleißner, A., Reislhuber, J.: 1-planarity of graphs with a rotation system. J. Graph Algorithms Appl. **19**(1), 67–86 (2015)
2. Bannister, M.J., Cabello, S., Eppstein, D.: Parameterized complexity of 1-planarity. In: Dehne, F., Solis-Oba, R., Sack, J.-R. (eds.) WADS 2013. LNCS, vol. 8037, pp. 97–108. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40104-6_9
3. Barahona, F.: On the computational complexity of Ising spin glass models. J. Phys. A Math. Gen. **15**(10), 3241 (1982)
4. Barahona, F.: Balancing signed toroidal graphs in polynomial time. Departamento de Matematicas, Universidad de Chile, Santiago, Chile (1983)
5. Barahona, F.: The max-cut problem on graphs not contractible to K5. Oper. Res. Lett. **2**(3), 107–111 (1983)
6. Barahona, F., Grötschel, M., Jünger, M., Reinelt, G.: An application of combinatorial optimization to statistical physics and circuit layout design. Oper. Res. **36**(3), 493–513 (1988)
7. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica **10**(1), 41–51 (1990)
8. De Simone, C., Diehl, M., Jünger, M., Mutzel, P., Reinelt, G., Rinaldi, G.: Exact ground states of Ising spin glasses: new experimental results with a branch-and-cut algorithm. J. Stat. Phys. **80**(1–2), 487–496 (1995)
9. Galluccio, A., Loebl, M.: Max cut in toroidal graphs. Instituto di Analisi dei Sistemi ed Informatica, Consiglio Nazionale delle Ricerche, Oktober 1998
10. Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J. ACM (JACM) **42**(6), 1115–1145 (1995)
11. Grötschel, M., Nemhauser, G.L.: A polynomial algorithm for the max-cut problem on graphs without long odd cycles. Math. Program. **29**(1), 28–40 (1984)
12. Grötschel, M., Pulleyblank, W.R.: Weakly bipartite graphs and the max-cut problem. Oper. Res. Lett. **1**(1), 23–27 (1981)
13. Hadlock, F.: Finding a maximum cut of a planar graph in polynomial time. SIAM J. Comput. **4**(3), 221–225 (1975)
14. Hopcroft, J.E., Tarjan, R.E.: Dividing a graph into triconnected components. SIAM J. Comput. **2**(3), 135–158 (1973)
15. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Proceedings of a symposium on the Complexity of Computer Computations, New York. The IBM Research Symposia Series, pp. 85–103. Plenum Press, New York (1972)
16. Korzhik, V.P., Mohar, B.: Minimal obstructions for 1-immersions and hardness of 1-planarity testing. J. Graph Theory **72**(1), 30–71 (2013)

17. Liers, F., Pardella, G.: A simple MAX-CUT algorithm for planar graphs. In: Cafieri, S., Mucherino, A., Nannicini, G., Tarissan, F., Liberti, L., (eds.) Proceedings of the 8th Cologne-Twente Workshop on Graphs and Combinatorial Optimization, CTW 2009, Paris, France, 2–4 June 2009, pp. 351–354 (2009)
18. Liers, F., Pardella, G.: Partitioning planar graphs: a fast combinatorial approach for max-cut. Comput. Optim. Appl. **51**(1), 323–344 (2012)
19. Mahajan, S., Ramesh, H.: Derandomizing approximation algorithms based on semidefinite programming. SIAM J. Comput. **28**(5), 1641–1663 (1999)
20. McGeoch, C.C.: Adiabatic Quantum Computation and Quantum Annealing: Theory and Practice. Synthesis Lectures on Quantum Computing. Morgan & Claypool Publishers, San Rafael (2014)
21. Mutzel, P.: Graphenalgorithmen, Master Vertiefungsvorlesung. Fakultät für Informatik, TU Dortmund (2016)
22. Orlova, G.I., Dorfman, Y.G.: Finding the maximum cut in a graph. Eng. Cybern. **10**(3), 502–506 (1972)
23. Pach, J., Tóth, G.: Graphs drawn with few crossings per edge. Combinatorica **17**(3), 427–439 (1997)
24. Papadimitriou, C.H., Yannakakis, M.: Optimization, approximation, and complexity classes. J. Comput. Syst. Sci. **43**(3), 425–440 (1991)
25. Shih, W.-K., Sun, W., Kuo, Y.-S.: Unifying maximum cut and minimum cut of a planar graph. IEEE Trans. Comput. **39**(5), 694–697 (1990)

Covering with Clubs: Complexity and Approximability

Riccardo Dondi[1], Giancarlo Mauri[2], Florian Sikora[3(✉)], and Italo Zoppis[2]

[1] Università degli Studi di Bergamo, Bergamo, Italy
riccardo.dondi@unibg.it
[2] Università degli Studi di Milano-Bicocca, Milan, Italy
{mauri,zoppis}@disco.unimib.it
[3] Université Paris-Dauphine, PSL Research University, CNRS UMR 7243, LAMSADE, 75016 Paris, France
florian.sikora@dauphine.fr

Abstract. Finding cohesive subgraphs in a network is a well-known problem in graph theory. Several alternative formulations of cohesive subgraph have been proposed, a notable example being s-club, which is a subgraph where each vertex is at distance at most s to the others. Here we consider the problem of covering a given graph with the minimum number of s-clubs. We study the computational and approximation complexity of this problem, when s is equal to 2 or 3. First, we show that deciding if there exists a cover of a graph with three 2-clubs is NP-complete, and that deciding if there exists a cover of a graph with two 3-clubs is NP-complete. Then, we consider the approximation complexity of covering a graph with the minimum number of 2-clubs and 3-clubs. We show that, given a graph $G = (V, E)$ to be covered, covering G with the minimum number of 2-clubs is not approximable within factor $O(|V|^{1/2-\varepsilon})$, for any $\varepsilon > 0$, and covering G with the minimum number of 3-clubs is not approximable within factor $O(|V|^{1-\varepsilon})$, for any $\varepsilon > 0$. On the positive side, we give an approximation algorithm of factor $2|V|^{1/2}\log^{3/2}|V|$ for covering a graph with the minimum number of 2-clubs.

1 Introduction

The quest for modules inside a network is a well-known and deeply studied problem in network analysis, with several application in different fields, like computational biology or social network analysis. A highly investigated problem is that of finding cohesive subgroups inside a network which in graph theory translates in highly connected subgraphs. A common approach is to look for cliques (i.e. complete graphs), and several combinatorial problems have been considered, notable examples being the Maximum Clique problem ([11, GT19]), the Minimum Clique Cover problem ([11, GT17]), and the Minimum Clique Partition problem ([11, GT15]). This last is a classical problem in theoretical computer science, whose goal is to partition the vertices of a graph into the minimum number of

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 153–164, 2018.
https://doi.org/10.1007/978-3-319-94667-2_13

cliques. The Minimum Clique Partition problem has been deeply studied since the seminal paper of Karp [15], studying its complexity in several graph classes [5,6,9,21].

In some cases, asking for a complete subgraph is too restrictive, as interesting highly connected graphs may have some missing edges due to noise in the data considered or because some pair may not be directly connected by an edge in the subgraph of interest. To overcome this limitation of the clique approach, alternative definitions of highly connected graphs have been proposed, leading to the concept of *relaxed clique* [16]. A relaxed clique is a graph $G = (V, E)$ whose vertices satisfy a property which is a relaxation of the clique property. Indeed, a clique is a subgraph whose vertices are all at distance one from each other and have the same degree (the size of the clique minus one). Different definitions of relaxed clique are obtained by modifying one of the properties of clique, thus leading to distance-based relaxed cliques, degree-based relaxed cliques, and so on (see for example [16]).

In this paper, we focus on a distance-based relaxation. In a clique all the vertices are required to be at distance at most one from each other. Here this constraint is relaxed, so that the vertices have to be at distance at most s, for an integer $s \geqslant 1$. A subgraph whose vertices are all distance at most s is called an *s-club* (notice that, when $s = 1$, an s-club is exactly a clique). The identification of s-clubs inside a network has been applied to social networks [1,18–20,23], and biological networks [3]. Interesting recent studies have shown the relevance of finding s-clubs in a network [18,20], in particular focusing on finding 2-clubs in real networks like DBLP or a European corporate network.

Contributions to the study of s-clubs mainly focus on the Maximum s-Club problem, that is the problem of finding an s-club of maximum size. Maximum s-Club is known to be NP-hard, for each $s \geqslant 1$ [4]. Even deciding whether there exists an s-club larger than a given size in a graph of diameter $s + 1$ is NP-complete, for each $s \geqslant 1$ [3]. The Maximum s-Club problem has been studied also in the approximability and parameterized complexity framework. A polynomial-time approximation algorithm with factor $|V|^{1/2}$ for every $s \geqslant 2$ on an input graph $G = (V, E)$ has been designed [2]. This is optimal, since the problem is not approximable within factor $|V|^{1/2-\varepsilon}$, on an input graph $G = (V, E)$, for each $\varepsilon > 0$ and $s \geqslant 2$ [2]. As for the parameterized complexity framework, the problem is known to be fixed-parameter tractable, when parameterized by the size of an s-club [7,17,22]. The Maximum s-Club problem has been investigated also for structural parameters and specific graph classes [12,13].

In this paper, we consider a different combinatorial problem, where we aim at covering the vertices of a network with a set of subgraphs. Similar to Minimum Clique Partition, we consider the problem of covering a graph with the minimum number of s-clubs such that each vertex belongs to an s-club. We denote this problem by Min s-Club Cover, and we focus in particular on the cases $s = 2$ and $s = 3$. We show some analogies and differences between Min s-Club Cover and Minimum Clique Partition. We start in Sect. 3 by considering the computational complexity of the problem of covering a graph with two or three s-clubs.

This is motivated by the fact that Clique Partition is known to be in P when we ask whether there exists a partition of the graph consisting of two cliques, while it is NP-hard to decide whether there exists a partition of the graph consisting of three cliques [10]. As for Clique Partition, we show that it is NP-complete to decide whether there exist three 2-clubs that cover a graph. On the other hand, we show that, unlike Clique Partition, it is NP-complete to decide whether there exist two 3-clubs that cover a graph. These two results imply also that Min 2-Club Cover and Min 3-Club Cover do not belong to the class XP for the parameter "number of clubs" in a cover.

Then, we consider the approximation complexity of Min 2-Club Cover and Min 3-Club Cover. We recall that, given an input graph $G = (V, E)$, Minimum Clique Partition is not approximable within factor $O(|V|^{1-\varepsilon})$, for any $\varepsilon > 0$, unless $P = NP$ [24]. Here we show that Min 2-Club Cover has a slightly different behavior, while Min 3-Club Cover is similar to Clique Partition. Indeed, in Sect. 4 we prove that Min 2-Club Cover is not approximable within factor $O(|V|^{1/2-\varepsilon})$, for any $\varepsilon > 0$, unless $P = NP$, while Min 3-Club Cover is not approximable within factor $O(|V|^{1-\varepsilon})$, for any $\varepsilon > 0$, unless $P = NP$. In Sect. 5, we present a greedy approximation algorithm that has factor $2|V|^{1/2}\log^{3/2}|V|$ for Min 2-Club Cover, which almost match the inapproximability result for the problem. We start the paper by giving in Sect. 2 some definitions and by formally defining the problem we are interested in. Some of the proofs (marked with ★) are omitted due to space constraint.

2 Preliminaries

Given a graph $G = (V, E)$ and a subset $V' \subseteq V$, we denote by $G[V']$ the subgraph of G induced by V'. Given two vertices $u, v \in V$, the distance between u and v in G, denoted by $d_G(u, v)$, is the length of a shortest path from u to v. The diameter of a graph $G = (V, E)$ is the maximum distance between two vertices of V. Given a graph $G = (V, E)$ and a vertex $v \in V$, we denote by $N_G(v)$ the set of neighbors of v, that is $N_G(v) = \{u : \{v, u\} \in E\}$. We denote by $N_G[v]$ the close neighborhood of V, that is $N_G[v] = N_G(v) \cup \{v\}$. Define $N^l_G(v) = \{u : u$ has distance at most l from $v\}$, with $1 \leqslant l \leqslant 2$. Given a set of vertices $X \subseteq V$ and l, with $1 \leqslant l \leqslant 2$, define $N^l_G(X) = \bigcup_{u \in X} N^l_G(u)$. We may omit the subscript G when it is clear from the context. Now, we give the definition of s-club, which is fundamental for the paper.

Definition 1. *Given a graph $G = (V, E)$, and a subset $V' \subseteq V$, $G[V']$ is an s-club if it has diameter at most s.*

Notice that an s-club must be a connected graph. We present now the formal definition of the Minimum s-Club Cover problem we are interested in.

Minimum s-Club Cover (Min s-Club Cover)
Input: a graph $G = (V, E)$ and an integer $s \geqslant 2$.
Output: a minimum cardinality collection $\mathcal{S} = \{V_1, \ldots, V_h\}$ such that, for each

i with $1 \leqslant i \leqslant h$, $V_i \subseteq V$, $G[V_i]$ is an s-club, and, for each vertex $v \in V$, there exists a set V_j, with $1 \leqslant j \leqslant h$, such that $v \in V_j$.

We denote by s-Club Cover(h), with $1 \leqslant h \leqslant |V|$, the decision version of Min s-Club Cover that asks whether there exists a cover of G consisting of at most h s-clubs.

Notice that while in Minimum Clique Partition we can assume that the cliques that cover a graph $G = (V, E)$ partition V, hence the cliques are vertex disjoint, we cannot make this assumption for Min s-Club Cover. Indeed, in a solution of Min s-Club Cover, a vertex may be covered by more than one s-club, in order to have a cover consisting of the minimum number of s-clubs. Consider the example of Fig. 1. The two 2-clubs induced by $\{v_1, v_2, v_3, v_4, v_5\}$ and $\{v_1, v_6, v_7, v_8, v_9\}$ cover G, and both these 2-clubs contain vertex v_1. However, if we ask for a partition of G, we need at least three 2-clubs. This difference between Minimum Clique Partition and Min s-Club Cover is due to the fact that, while being a clique is a hereditary property, this is not the case for being an s-club. If a graph G is an s-club, then a subgraph of G may not be an s-club (for example a star is a 2-club, but the subgraph obtained by removing its center is not anymore a 2-club).

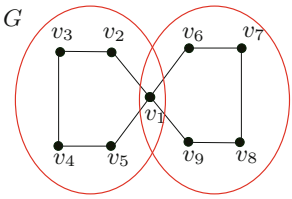


Fig. 1. A graph G and a cover consisting of two 2-clubs (induced by the vertices in the ovals). Notice that the 2-clubs of this cover must both contain vertex v_1.

3 Computational Complexity

In this section we investigate the computational complexity of 2-Club Cover and 3-Club Cover and we show that 2-Club Cover(3), that is deciding whether there exists a cover of a graph G with three 2-clubs, and 3-Club Cover(2), that is deciding whether there exists a cover of a graph G with two 3-clubs, are NP-complete.

3.1 2-Club Cover(3) is NP-Complete

In this section we show that 2-Club Cover(3) is NP-complete by giving a reduction from the 3-Clique Partition problem, that is the problem of computing whether there exists a partition of a graph $G^p = (V^p, E^p)$ in three cliques. Consider

an instance $G^p = (V^p, E^p)$ of 3-Clique Partition, we construct an instance $G = (V, E)$ of 2-Club Cover(3) (see Fig. 2). The vertex set V is defined as follows:

$$V = \{w_i : v_i \in V^p\} \cup \{w_{i,j} : \{v_i, v_j\} \in E^p \wedge i < j\}$$

The set E of edges is defined as follows:

$$E = \{\{w_i, w_{i,j}\}, \{w_i, w_{h,i}\} : v_i \in V^p, w_i, w_{i,j}, w_{h,i} \in V\}$$
$$\cup \{\{w_{i,j}, w_{i,l}\}, \{w_{i,j}, w_{h,i}\}, \{w_{h,i}, w_{z,i}\} : w_{i,j}, w_{i,l}, w_{h,i}, w_{z,i} \in V\}$$

Before giving the main results of this section, we prove a property of G.

Lemma 2 (★). *Let $G^p = (V^p, E^p)$ be an instance of* 3-Clique Partition *and let $G = (V, E)$ be the corresponding instance of* 2-Club Cover(3)*. Then, given two vertices $v_i, v_j \in V^p$ and the corresponding vertices $w_i, w_j \in V$:*

- *if $\{v_i, v_j\} \in E^p$, then $d_G(w_i, w_j) = 2$*
- *if $\{v_i, v_j\} \notin E^p$, then $d_G(w_i, w_j) \geqslant 3$*

We are now able to prove the main properties of the reduction.

Lemma 3. *Let $G^p = (V^p, E^p)$ be a graph input of* 3-Clique Partition *and let $G = (V, E)$ be the corresponding instance of* 2-Club Cover(3)*. Then, given a solution of* 3-Clique Partition *on $G^p = (V^p, E^p)$, we can compute in polynomial time a solution of* 2-Club Cover(3) *on $G = (V, E)$.*

Proof. Consider a solution of 3-Clique Partition on $G^p = (V^p, E^p)$, and let V_1^p, V_2^p, $V_3^p \subseteq V^p$ be the sets of vertices of G^p that partition V^p. We define a solution of 2-Club Cover(3) on $G = (V, E)$ as follows. For each d, with $1 \leqslant d \leqslant 3$, define

$$V_d = \{w_j \in V : v_j \in V_d^p\} \cup \{w_{i,j} : v_i \in V_d^p\}$$

We show that each $G[V_d]$, with $1 \leqslant d \leqslant 3$, is a 2-club. Consider two vertices $w_i, w_j \in V_d$, with $1 \leqslant i < j \leqslant |V|$. Since they correspond to two vertices $v_i, v_j \in V^p$ that belong to a clique of G^p, it follows that $\{v_i, v_j\} \in E^p$ and $w_{i,j} \in V_d$. Thus $d_{G[V_d]}(w_i, w_j) = 2$. Now, consider the vertices $w_i \in V_d$, with $1 \leqslant i \leqslant |V|$, and $w_{h,z} \in V_d$, with $1 \leqslant h < z \leqslant |V|$. If $i = h$ or $i = z$, assume w.l.o.g. $i = h$, then by construction $d_{G[V_d]}(w_i, w_{i,z}) = 1$. Assume that $i \neq h$ and $i \neq z$ (assume w.l.o.g. that $i < h < z$), since $w_{h,z} \in V_d$, it follows that $w_h \in V_d$. Since $w_i, w_h \in V_d$, it follows that $w_{i,h} \in V_d$. By construction, there exist edges $\{w_{i,h}, w_{h,z}\}$, $\{w_i, w_{i,h}\}$ in E^p, thus implying that $d_{G[V_d]}(w_i, w_{h,z}) = 2$. Finally, consider two vertices $w_{i,j}, w_{h,z} \in V_d$, with $1 \leqslant i < j \leqslant |V|$ and $1 \leqslant h < z \leqslant |V|$. Then, by construction, $w_i \in V_d$ and $w_h \in V_d$. But then, $w_{i,h}$ belongs to V_d, and, by construction, $\{w_{i,j}, w_{i,h}\} \in E$ and $\{w_{h,z}, w_{i,h}\} \in E$. It follows that $d_{G[V_d]}(w_{i,j}, w_{h,z}) = 2$.

We conclude the proof observing that, by construction, since V_1^p, V_2^p, V_3^p partition V^p, it holds that $V = V_1 \cup V_2 \cup V_3$, thus $G[V_1]$, $G[V_2]$, $G[V_3]$ covers G. □

Based on Lemma 2, we can prove the following result.

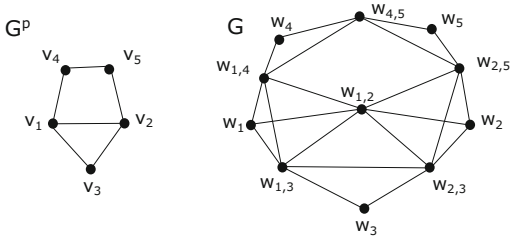


Fig. 2. An example of a graph G^p input of 3-Clique Partition and the corresponding graph G input of 2-Club Cover(3).

Lemma 4 (★). *Let $G^p = (V^p, E^p)$ be a graph input of* 3-Clique Partition *and let $G = (V, E)$ be the corresponding instance of* 2-Club Cover(3)*. Then, given a solution of* 2-Club Cover(3) *on $G = (V, E)$, we can compute in polynomial time a solution of* 3-Clique Partition *on $G^p = (V^p, E^p)$.*

Now, we can prove the main result of this section.

Theorem 5 (★). 2-Club Cover(3) *is NP-complete.*

3.2 3-Club Cover(2) is NP-Complete

In this section we show that 3-Club Cover(2) is NP-complete by giving a reduction from a variant of Sat called 5-Double-Sat. Recall that a literal is positive if it is a non-negated variable, while it is negative if it is a negated variable.

Given a collection of clauses $\mathcal{C} = \{C_1, \ldots, C_p\}$ over the set of variables $X = \{x_1, \ldots, x_q\}$, where each $C_i \in \mathcal{C}$, with $1 \leqslant i \leqslant p$, contains exactly five literals and does not contain both a variable and its negation, 5-Double-Sat asks for a truth assignment to the variables in X such that each clause C_i, with $1 \leqslant i \leqslant p$, is *double-satisfied*. A clause C_i is double-satisfied by a truth assignment f to the variables X if there exist a positive literal and a negative literal in C_i that are both satisfied by f. Notice that we assume that there exist at least one positive literal and at least one negative literal in each clause C_i, with $1 \leqslant i \leqslant p$, otherwise C_i cannot be doubled-satisfied. Moreover, we assume that each variable in an instance of 5-Double-Sat appears both as a positive literal and a negative literal in the instance. Notice that if this is not the case, for example a variable appears only as a positive literal, we can assign a true value to the variable, as defining an assignment to false does not contribute to double-satisfy any clause. First, we show that 5-Double-Sat is NP-complete, which may be of independent interest.

Theorem 6 (★). 5-Double-Sat *is NP-complete.*

Let us now give the construction of the reduction from 5-Double-Sat to 3-Club Cover(2). Consider an instance of 5-Double-Sat consisting of a set $\mathcal{C}$ of clauses $C_1, \ldots, C_p$ over set $X = \{x_1, \ldots, x_q\}$ of variables. We assume that it is

not possible to double-satisfy all the clauses by setting at most two variables to true or to false (this can be easily checked in polynomial-time).

Before giving the details, we present an overview of the reduction. Given an instance $(X, \mathcal{C})$ of 5-Double-Sat, for each positive literal x_i, with $1 \leqslant i \leqslant q$, we define vertices $x^T_{i,1}$, $x^T_{i,2}$ and for each negative literal $\overline{x_i}$, with $1 \leqslant i \leqslant q$, we define a vertex x^F_i. Moreover, for each clause $C_j \in \mathcal{C}$, with $1 \leqslant j \leqslant p$, we define a vertex $v_{C,j}$. We define other vertices to ensure that some vertices have distance not greater than three and to force the membership to one of the two 3-clubs of the solution (see Lemma 7). The construction implies that for each i with $1 \leqslant i \leqslant q$, $x^T_{i,1}$ and x^F_i belong to different 3-clubs (see Lemma 8); this corresponds to a truth assignment to the variables in X. Then, we are able to show that each vertex $v_{C,j}$ belongs to the same 3-club of a vertex $x^T_{i,1}$, with $1 \leqslant i \leqslant q$, and of a vertex x^F_h, with $1 \leqslant h \leqslant q$, adjacent to $v_{C,j}$ (see Lemma 10); these vertices correspond to a positive literal x_i and a negative literal $\overline{x_h}$, respectively, that are satisfied by a truth assignment, hence C_j is double-satisfied.

Now, we give the details of the reduction. Let $(X, \mathcal{C})$ be an instance of 5-Double-Sat, we construct an instance $G = (V, E)$ of 3-Club Cover(2) as follows (see Fig. 3). The vertex set V is defined as follows:

$$V = \{r, r', r_T, r'_T, r^*_T, r_F, r'_F\} \cup \{x^T_{i,1}, x^T_{i,2}, x^F_i : x_i \in X\} \cup \{v_{C,j} : C_j \in \mathcal{C}\} \cup \{y_1, y_2, y\}$$

The edge set E is defined as follows:

$$\begin{aligned}
E = {} & \{\{r, r'\}, \{\{r', r_T\}, \{r', r^*_T\}\{r', r_F\}\} \cup \{\{r_T, x^T_{i,1}\} : x_i \in X\} \\
& \cup \{\{r_F, x^F_i\} : x_i \in X\} \cup \{\{r'_T, x^T_{i,1}\} : x_i \in X\} \cup \{\{r'_F, x^F_i\} : x_i \in X\} \\
& \cup \{\{x^T_{i,1}, x^T_{i,2}\} : x_i \in X\} \cup \{\{r^*_T, x^T_{i,2}\}, \{y_1, x^T_{i,2}\} : x_i \in X\} \\
& \cup \{\{x^T_{i,2}, x^F_j\} : x_i, x_j \in X, i \neq j\} \cup \{\{x^T_{i,1}, v_{C,j}\} : x_i \in C_j\} \cup \{\{x^F_i, v_{C,j}\} : \overline{x_i} \in C_j\} \\
& \cup \{\{v_{C,j}, y\} : C_j \in \mathcal{C}\} \cup \{\{y, y_2\}, \{y_1, y_2\}, \{y_1, r'_T\}, \{y_1, r'_F\}\}
\end{aligned}$$

We start by proving some properties of the graph G.

Lemma 7 (★). *Consider an instance $(\mathcal{C}, X)$ of* 5-Double-Sat *and let $G = (V, E)$ be the corresponding instance of* 3-Club Cover(2)*. Then, (1) $d_G(r', y) > 3$, (2) $d_G(r, y) > 3$, (3) $d_G(r, v_{C,j}) > 3$, for each j with $1 \leqslant j \leqslant p$, and (4) $d_G(r, r'_F) > 3$, $d_G(r, r'_T) > 3$.*

Consider two sets $V_1 \subseteq V$ and $V_2 \subseteq V$, such that $G[V_1]$ and $G[V_2]$ are two 3-clubs of G that cover G. As a consequence of Lemma 7, it follows that r and r' are in exactly one of $G[V_1]$, $G[V_2]$, w.l.o.g. $G[V_1]$, while r'_T, r'_F, y and $v_{C,j}$, for each j with $1 \leqslant j \leqslant p$, belong to $G[V_2]$ and not to $G[V_1]$.

Next, we show a crucial property of the graph G built by the reduction.

Lemma 8 (★). *Given an instance $(\mathcal{C}, X)$ of* 5-Double-Sat*, let $G = (V, E)$ be the corresponding instance of* 3-Club Cover(2)*. Then, for each i with $1 \leqslant i \leqslant q$, $d_G(x^T_{i,1}, x^F_i) > 3$.*

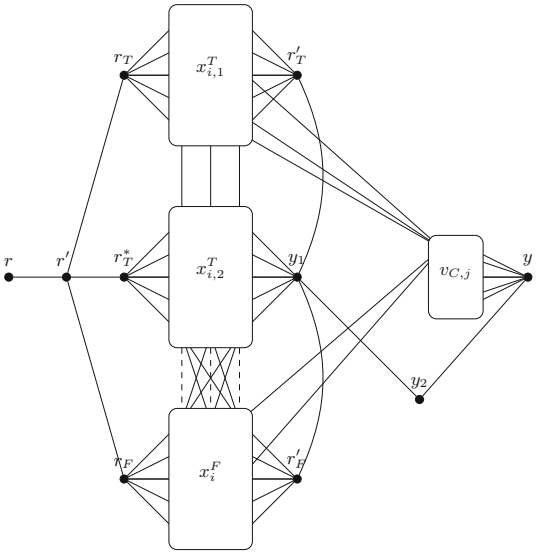


Fig. 3. Schematic construction for the reduction from 5-Double-Sat to 3-Club Cover(2).

Now, we are able to prove the main results of this section.

Lemma 9 (★). *Given an instance* $(\mathcal{C}, X)$ *of* 5-Double-Sat*, let* $G = (V, E)$ *be the corresponding instance of* 3-Club Cover(2)*. Then, given a truth assignment that double-satisfies* $\mathcal{C}$*, we can compute in polynomial-time two* 3*-clubs that cover* G*.*

Lemma 10. *Given an instance* $(\mathcal{C}, X)$ *of* 5-Double-Sat*, let* $G = (V, E)$ *be the corresponding instance of* 3-Club Cover(2)*. Then, given two* 3*-clubs that cover* G*, we can compute in polynomial time a truth assignment that double-satisfies* $\mathcal{C}$*.*

Proof. Consider two 3-clubs $G[V_1]$, $G[V_2]$, with $V_1, V_2 \subseteq V$, that cover G. First, notice that by Lemma 7 we assume that $r, r' \in V_1 \setminus V_2$, while $y, r'_T, r'_F \in V_2 \setminus V_1$ and $v_{C,j} \in V_2 \setminus V_1$, for each j with $1 \leqslant j \leqslant p$. Moreover, by Lemma 8 it follows that for each i with $1 \leqslant i \leqslant q$, $x^T_{i,1}$ and x^F_i do not belong to the same 3-club, that is exactly one belongs to V_1 and exactly one belongs to V_2.

By construction, each path of length at most three from a vertex $v_{C,j}$, with $1 \leqslant j \leqslant p$, to r'_F must pass through some x^F_h, with $1 \leqslant h \leqslant q$. Similarly, each path of length at most three from a vertex $v_{C,j}$, with $1 \leqslant j \leqslant p$, to r'_T must pass through some $x^T_{i,1}$. Assume that $v_{C,j}$, with $1 \leqslant j \leqslant p$, is not adjacent to a vertex $x^T_{i,1} \in V_2$, with $1 \leqslant i \leqslant q$ ($x^F_h \in V_2$, with $1 \leqslant h \leqslant p$ respectively). It follows that $v_{C,j}$ is only adjacent to y and to vertices x^F_w, with $1 \leqslant w \leqslant q$ ($x^T_{u,1}$, with $1 \leqslant u \leqslant q$, respectively) in $G[V_2]$. In the first case, notice that y is adjacent only to $v_{C,z}$, with $1 \leqslant z \leqslant p$, and y_2, none of which is adjacent to r'_T (r'_F, respectively), thus implying that this path from $v_{C,j}$ to r'_T (to r'_F, respectively) has length at least 4. In the second case, x^F_w ($x^T_{u,1}$, respectively) is adjacent to

r'_F, r_F, $v_{C,j}$ and $x^T_{i,2}$ (r'_T, r_T, $v_{C,j}$, $x^T_{u,2}$, respectively), none of which is adjacent to r'_T (r'_F, respectively), implying that also in this case the path from $v_{C,j}$ to r'_T (to r'_F, respectively) has length at least 4. Since $r'_T, r'_F, v_{C,j} \in V_2$, it follows that, for each $v_{C,j}$, the set V_2 contains a vertex $x^T_{i,1}$, with $1 \leqslant i \leqslant q$, and a vertex x^F_h, with $1 \leqslant h \leqslant q$, connected to $v_{C,j}$.

By Lemma 8 exactly one of $x^T_{i,1}$, x^F_i belongs to V_2, thus we can construct a truth assignment f as follows: $f(x_i) :=$ true, if $x^T_{i,1} \in V_2$, $f(x_i) :=$ false, if $x^F_i \in V_2$. The assignment f double-satisfies each clause of $\mathcal{C}$, since each $v_{C,j}$ is connected to a vertex $x^T_{i,1}$, for some i with $1 \leqslant i \leqslant q$, and a vertex x^F_h, for some h with $1 \leqslant h \leqslant q$. □

Based on Lemmas 9 and 10, and on the NP-completeness of 5-Double-Sat (see Theorem 6), we can conclude that 3-Club Cover(2) is NP-complete.

Theorem 11 (★). 3-Club Cover(2) *is NP-complete.*

4 Hardness of Approximation

In this section we consider the approximation complexity of Min 2-Club Cover and Min 3-Club Cover and we prove that Min 2-Club Cover is not approximable within factor $O(|V|^{1/2-\varepsilon})$, for each $\varepsilon > 0$, and that Min 3-Club Cover is not approximable within factor $O(|V|^{1-\varepsilon})$, for each $\varepsilon > 0$. The proof for Min 2-Club Cover is obtained with a reduction very similar to that of Sect. 3.1, except from the fact that we reduce Minimum Clique Partition to Min 2-Club Cover.

Corollary 12 (★). *Unless* $P = NP$, Min 2-Club Cover *is not approximable within factor* $O(|V|^{1/2-\varepsilon})$, *for each* $\varepsilon > 0$.

Next, we show that Min 3-Club Cover is not approximable within factor $O(|V|^{1-\varepsilon})$, for each $\varepsilon > 0$, unless $P = NP$, by giving a preserving-factor reduction from Minimum Clique Partition.

Consider an instance $G^p = (V^p, E^p)$ of Minimum Clique Partition, we construct an instance $G = (V, E)$ of Min 3-Club Cover by adding a pendant vertex connected to each vertex of V^p. Formally, $V = \{u_i, w_i : v_i \in V^p\}$, $E = \{\{u_i, w_i\} : 1 \leqslant i \leqslant |V^p|\} \cup \{\{u_i, u_j\} : \{v_i, v_j\} \in E^p\}\}$.

We prove now the main properties of the reduction.

Lemma 13 (★). *Let* $G^p = (V^p, E^p)$ *be an instance of* Minimum Clique Partition *and let* $G = (V, E)$ *be the corresponding instance of* Min 3-Club Cover. *Then, given a solution of* Minimum Clique Partition *on* $G^p = (V^p, E^p)$ *consisting of* k *cliques, we can compute in polynomial time a solution of* Min 3-Club Cover *on* $G = (V, E)$ *consisting of* k 3*-clubs.*

Lemma 14 (★). *Let* $G^p = (V^p, E^p)$ *be a graph input of* Minimum Clique Partition *and let* $G = (V, E)$ *be the corresponding instance of* Min 3-Club Cover. *Then, given a solution of* Min 3-Club Cover *on* $G = (V, E)$ *consisting of* k 3*-clubs, we can compute in polynomial time a solution of* Minimum Clique Partition *on* $G^p = (V^p, E^p)$ *consisting of* k *cliques.*

Lemmas 13 and 14 imply the following result.

Theorem 15 (★). Min 3-Club Cover *is not approximable within factor* $O(|V|^{1-\varepsilon})$*, for each* $\varepsilon > 0$*, unless* $P = NP$*.*

5 An Approximation Algorithm for Min 2-Club Cover

In this section, we present an approximation algorithm for Min 2-Club Cover that achieves an approximation factor of $2|V|^{1/2}\log^{3/2}|V|$. Notice that, due to the result in Sect. 4, the approximation factor is almost tight. We start by describing the approximation algorithm, then we present the analysis of the approximation factor.

Algorithm 1. Club-Cover-Approx

Data: a graph G
Result: a cover $\mathcal{S}$ of G
$V' := V$; /* V' is the set of uncovered vertices of G, initialized to V */
$\mathcal{S} := \emptyset$;
while $V' \neq \emptyset$ **do**
  Let v be a vertex of V such that $|N[v] \cap V'|$ is maximum;
  Add $N[v]$ to $\mathcal{S}$;
  $V' := V' \setminus N[v]$;

Club-Cover-Approx is similar to the greedy approximation algorithm for Minimum Dominating Set and Minimum Set Cover. While there exists an uncovered vertex of G, the Club-Cover-Approx algorithm greedily defines a 2-club induced by the set $N[v]$ of vertices, with $v \in V$, such that $N[v]$ covers the maximum number of uncovered vertices (notice that some of the vertices of $N[v]$ may already be covered). While for Minimum Dominating Set the choice of each iteration is optimal, here the choice is suboptimal. Notice that indeed computing a maximum 2-club is NP-hard.

Clearly the algorithm returns a feasible solution for Min 2-Club Cover, as each set $N[v]$ picked by the algorithm is a 2-club and, by construction, each vertex of V is covered. Next, we show the approximation factor yielded by the Club-Cover-Approx algorithm for Min 2-Club Cover.

First, consider the set V_D of vertices $v \in V$ picked by the Club-Cover-Approx algorithm, so that $N[v]$ is added to $\mathcal{S}$. Notice that $|V_D| = |\mathcal{S}|$ and that V_D is a dominating set of G, since, at each step, the vertex v picked by the algorithm dominates each vertex in $N[v]$, and each vertex in V is covered by the algorithm, so it belongs to some $N[v]$, with $v \in V_D$.

Let D be a minimum dominating set of the input graph G. By the property of the greedy approximation algorithm for Minimum Dominating Set, the set V_D has the following property [14]:

$$|V_D| \leqslant |D| \log |V| \tag{1}$$

The size of a minimum dominating set in graphs of diameter bounded by 2 (hence 2-clubs) has been considered in [8], where the following result is proven.

Lemma 16 ([8]). *Let $H = (V_H, E_H)$ be a 2-club, then H has a dominating set of size at most $1 + \sqrt{|V_H| + \ln(|V_H|)}$.*

The approximation factor $2|V|^{1/2}\log^{3/2}|V|$ for Club-Cover-Approx is obtained by combining Lemma 16 and Eq. 1.

Theorem 17. *Let OPT be an optimal solution of* Min 2-Club Cover*, then Club-Cover-Approx returns a solution having at most $2|V|^{1/2}\log^{3/2}|V||OPT|$ 2-clubs.*

Proof. Let D be a minimum dominating set of G and let OPT be an optimal solution of Min 2-Club Cover. We start by proving that $|D| \leqslant 2|OPT||V|^{1/2}\log^{1/2}|V|$. For each 2-club $G[C]$, with $C \subseteq V$, that belongs to OPT, by Lemma 16 there exists a dominating set D_C of size at most $1 + \sqrt{|C| + \ln(|C|)} \leqslant 2\sqrt{|C| + \ln(|C|)}$. Since $|C| \leqslant |V|$, it follows that each 2-club $G[C]$ that belongs to OPT has a dominating set of size at most $2\sqrt{|V| + \ln(|V|)}$. Consider $D' = \bigcup_{C \in OPT} D_C$. It follows that D' is a dominating set of G, since the 2-clubs in OPT covers G. Since D' contains $|OPT|$ sets D_C and $|D_C| \leqslant 2\sqrt{|V| + \ln(|V|)}$, for each $G[C] \in OPT$, it follows that $|D'| \leqslant 2|OPT|\sqrt{|V| + \ln(|V|)}$. Since D is a minimum dominating set, it follows that $|D| \leqslant |D'| \leqslant 2|OPT|(\sqrt{|V| + \ln(|V|)})$. By Eq. 1, it holds $|V_D| \leqslant 2|D|\log|V|$ thus $|V_D| \leqslant 2|V|^{1/2}\ln^{1/2}|V|\log|V||OPT| \leqslant 2|V|^{1/2}\log^{3/2}|V||OPT|$. □

6 Conclusion

There are some interesting direction for the problem of covering a graph with s-clubs. From the computational complexity point of view, the main open problem is whether 2-Club Cover(2) is NP-complete or is in P. Moreover, it would be interesting to study the computational/parameterized complexity of the problem in specific graph classes, as done for Minimum Clique Partition [5,6,9,21].

References

1. Alba, R.D.: A graph-theoretic definition of a sociometric clique. J. Math. Sociol. **3**, 113–126 (1973)
2. Asahiro, Y., Doi, Y., Miyano, E., Samizo, K., Shimizu, H.: Optimal approximation algorithms for maximum distance-bounded subgraph problems. Algorithmica **80**, 1834–1856 (2017)
3. Balasundaram, B., Butenko, S., Trukhanov, S.: Novel approaches for analyzing biological networks. J. Comb. Optim. **10**(1), 23–39 (2005)
4. Bourjolly, J., Laporte, G., Pesant, G.: An exact algorithm for the maximum k-club problem in an undirected graph. Eur. J. Oper. Res. **138**(1), 21–28 (2002)
5. Cerioli, M.R., Faria, L., Ferreira, T.O., Martinhon, C.A.J., Protti, F., Reed, B.A.: Partition into cliques for cubic graphs: planar case, complexity and approximation. Discrete Appl. Math. **156**(12), 2270–2278 (2008)
6. Cerioli, M.R., Faria, L., Ferreira, T.O., Protti, F.: A note on maximum independent sets and minimum clique partitions in unit disk graphs and penny graphs: complexity and approximation. RAIRO Theor. Inf. Appl. **45**(3), 331–346 (2011)

7. Chang, M., Hung, L., Lin, C., Su, P.: Finding large k-clubs in undirected graphs. Computing **95**(9), 739–758 (2013)
8. Desormeaux, W.J., Haynes, T.W., Henning, M.A., Yeo, A.: Total domination in graphs with diameter 2. J. Graph Theory **75**(1), 91–103 (2014)
9. Dumitrescu, A., Pach, J.: Minimum clique partition in unit disk graphs. Graphs Comb. **27**(3), 399–411 (2011)
10. Garey, M.R., Johnson, D.S., Stockmeyer, L.J.: Some simplified NP-complete graph problems. Theor. Comput. Sci. **1**(3), 237–267 (1976)
11. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
12. Golovach, P.A., Heggernes, P., Kratsch, D., Rafiey, A.: Finding clubs in graph classes. Discrete Appl. Math. **174**, 57–65 (2014)
13. Hartung, S., Komusiewicz, C., Nichterlein, A.: Parameterized algorithmics and computational experiments for finding 2-clubs. J. Graph Algorithms Appl. **19**(1), 155–190 (2015)
14. Johnson, D.S.: Approximation algorithms for combinatorial problems. J. Comput. Syst. Sci. **9**(3), 256–278 (1974)
15. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Proceedings of a symposium on the Complexity of Computer Computations, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 20–22 March 1972. The IBM Research Symposia Series, pp. 85–103. Plenum Press, New York (1972)
16. Komusiewicz, C.: Multivariate algorithmics for finding cohesive subnetworks. Algorithms **9**(1), 21 (2016)
17. Komusiewicz, C., Sorge, M.: An algorithmic framework for fixed-cardinality optimization in sparse graphs applied to dense subgraph problems. Discrete Appl. Math. **193**, 145–161 (2015)
18. Laan, S., Marx, M., Mokken, R.J.: Close communities in social networks: boroughs and 2-clubs. Social Netw. Anal. Min. **6**(1), 20:1–20:16 (2016)
19. Mokken, R.: Cliques, clubs and clans. Qual. Quant. Int. J. Methodol. **13**(2), 161–173 (1979)
20. Mokken, R.J., Heemskerk, E.M., Laan, S.: Close communication and 2-clubs in corporate networks: Europe 2010. Social Netw. Anal. Min. **6**(1), 40:1–40:19 (2016)
21. Pirwani, I.A., Salavatipour, M.R.: A weakly robust PTAS for minimum clique partition in unit disk graphs. Algorithmica **62**(3–4), 1050–1072 (2012)
22. Schäfer, A., Komusiewicz, C., Moser, H., Niedermeier, R.: Parameterized computational complexity of finding small-diameter subgraphs. Optim. Lett. **6**(5), 883–891 (2012)
23. Zoppis, I., Dondi, R., Santoro, E., Castelnuovo, G., Sicurello, F., Mauri, G.: Optimizing social interaction - a computational approach to support patient engagement. In: Zwiggelaar, R., Gamboa, H., Fred, A.L.N., i Badia, S.B. (eds.) Proceedings of the 11th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2018) - Volume 5: HEALTHINF, Funchal, Madeira, Portugal, 19–21 January 2018, pp. 651–657. SciTePress (2018)
24. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. Theory Comput. **3**(1), 103–128 (2007)

On the Expected Number of Distinct Gapped Palindromic Factors

Philippe Duchon[1,2] and Cyril Nicaud[3(✉)]

[1] Univ. 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. An α-gapped palindromic factor of a word is a factor of the form $uv\overline{u}$, where $\overline{u}$ is the reversal of u and where $|uv| \leq \alpha|u|$ for some fixed $\alpha \geq 1$. We give an asymptotic estimate of the expected number of distinct palindromic factors in a random word for a memoryless source, where each letter is generated independently from the other, according to some fixed probability distribution on the alphabet.

1 Introduction

An α-gapped palindrome is a word of the form $uv\overline{u}$, where $\overline{u}$ is the reversal[1] of u and where $|uv| \leq \alpha|u|$, for some fixed $\alpha \geq 1$. Initially motivated by applications to bioinformatics, several articles in the literature focus on studying the α-gapped palindromic factors that occur in a given word [5,7]. Different directions were taken in these studies, and it is now known that there are at most a linear number of distinct α-gapped palindromic factors in a word [1,5], and that they can be computed in linear time [6,8,13].

In this paper, we are interested in the probabilistic properties related to this notion: if w is a random word of length n, what can be said about its α-gapped palindromic factors? To answer this kind of question, the probabilistic model must be specified. In the sequel, we will consider words generated using a *memoryless source*: each letter is chosen independently from each other, following a fixed probability distribution on the alphabet. Together with C. Pivoteau, we gave several results in [4]: the expected number of α-gapped palindromic factors and the expected length of the longest such factor. These were obtained using classical techniques from analytic combinatorics, together with elementary discrete probabilities. We also adapted, almost readily, a result by Rubinchik and Shur [11] on the expected number of *distinct* palindromic factors when the distribution is the *uniform distribution*, to *distinct* α-gapped palindromic factors. But this technique fails to work when the distribution is not uniform.

We aim at completing the works [4,11] by studying the number of distinct α-gapped palindromic factors in a random word generated by a memoryless

[1] The reversal of $u = u_1 \cdots u_n$ is $\overline{u} = u_n \cdots u_1$.

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 165–176, 2018.
https://doi.org/10.1007/978-3-319-94667-2_14

source. Beside the combinatorial and probabilistic motivations, note that knowledge on the typical number of distinct factors can be useful in the design of data structures, as it can give hints on the likely memory size needed for a typical input. For instance, the number of vertices of the graph EERTREE introduced in [12] is the number of distinct palindromic factors, which is in $\Theta(\sqrt{n})$ in expectation for the uniform distribution [11].

The classical techniques we used in [4] are not well suited to handle distinct factors. This is why we propose in [3] a probabilistic process that focuses on the notion of distinctness, in order to develop useful methodologies. The process we studied is the following: generate N random words of length L, independently, and remove duplicates. What does the resulting random set S looks like? We gave a precise characterization of the typical composition of letters of a word of S. More than the result itself, the techniques we used, based on classical analysis of functions with several variables, can be used to try to tackle other questions involving distinctness and non-uniform models. It also hints that uniform distributions are really singular, hiding some complicated situations that appear for non-uniform distribution only.

In this article, we use techniques that are similar to those introduced in [3] in order to estimate the expected number of distinct α-gapped palindromic factors in a random word, generated by a memoryless source. Our problem is more difficult than [3] for two main reasons: factors of a random word are not independent and α-gapped palindromic factors have various lengths. As we will see, these technical difficulties can be overcome, and we give in the sequel two main results: an estimate of the expected number of distinct α-gapped palindromic factors and a description of the factors that are more likely to occur, in term of their lengths and of their composition of letters.

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 $\|\mathbf{x}\| = \sqrt{\sum_{i\in[k]} x_i^2}$ denote the Euclidean norm. A vector $\mathbf{x} = (x_1, \dots, x_k) \in \mathbb{R}^k$ is a *probability vector* if every $x_i \in [0, 1]$ and $\sum_{i=1}^k x_i = 1$.

Words. Let $A = \{a_1, \dots, a_k\}$ be an alphabet with $k \geq 2$ letters. Throughout the article, the alphabet A, and therefore k, are fixed. We denote the empty word by ε. If $u = u_1 \cdots u_n$ is a word of length n on A, then its *reversal* is the word $\overline{u} = u_n \cdots u_1$. A *palindrome* is a word of the form $u\lambda\overline{u}$ where $\lambda \in A \cup \{\varepsilon\}$, that is, where λ is either a letter or empty. Let $\alpha \geq 1$ be a real number. An *α-gapped* palindrome is a word of the form $uv\overline{u}$ where $|uv| \leq \alpha|u|$.

For any word $w \in A^*$, the *composition vector* (or *Parikh vector*) of w is the vector $(|w|_1, \dots, |w|_k)$, where $|w|_i$ is the number of occurrences of a_i in w. If w is not empty, its *frequency vector* is the probability vector $(\frac{|w|_1}{|w|}, \dots, \frac{|w|_k}{|w|})$. We let $\mathcal{W}_m(\mathbf{x})$ denote the set of words of length m with frequency vector $\mathbf{x}$ (which is empty if $m\mathbf{x}$ does not have nonnegative integer coordinates).

Probabilities. Throughout the article, we assume some probability vector $\mathbf{p} = (p_1, \ldots, p_k)$ 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 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 $p_{\max} = \max_{i \in [k]} p_i < 1$ denote the maximal value of $\mathbf{p}$.

We will use the (natural-based) entropy function [9] on k positive variables, which is defined by $H(\mathbf{x}) = H(x_1, \ldots, x_k) = -\sum_{i=1}^{k} x_i \log x_i$. We also borrow the function Φ from [3], defined for any non-negative t by $\Phi(t) = \sum_{i=1}^{k} p_i^t$.

3 Expected Number of Distinct α-Gapped Factors

Recall that the $\tilde{\Theta}$ notation means "asymptotically of the same growth, up to some polylogarithmic multiplicative factors". More precisely, a positive sequence $(u_n)_{n \geq 0}$ is $\tilde{\Theta}(n^d)$ if there exists $\delta > 0$ such that $n^d (\log n)^{-\delta} \leq u_n \leq n^d (\log n)^{\delta}$, for n sufficiently large. It is in the same vein, though a bit more precise, as saying that for all $\varepsilon > 0$, u_n is $\mathcal{O}(n^{d+\varepsilon})$ and u_n is $\Omega(n^{d-\varepsilon})$. Our main result is the following.

Theorem 1. *Let $\alpha \geq 1$. The expected number of distinct α-gapped palindromic factors is $\tilde{\Theta}(n^{c^*})$, where $c^* \in (0,1)$ is the unique positive solution of the equation*

$$\Phi(2c)\Phi(c)^{\alpha-1} = 1.$$

Observe that, even if we assume that $\mathbf{p}$ is not the uniform distribution in this paper, the result is compatible with the estimations of [3,11]: we have $\Phi(t) = k^{1-t}$ for the uniform distribution, so the equation rewrites $k^{1-2c+(\alpha-1)(1-c)=1} = 1$, so that $c^* = \frac{\alpha}{\alpha+1}$. The value of c^* for non-uniform $\mathbf{p}$ are depicted in Fig. 1.

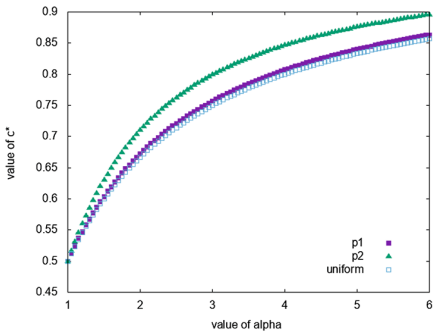


Fig. 1. The values of c^* as α ranges from 1 to 6, for three different probability vectors on an alphabet with three letters ($k = 3$): $\overline{\mathbf{x}}$ the uniform distribution plotted with empty squares, $p_1 = (\frac{1}{2}, \frac{1}{3}, \frac{1}{6})$ plotted with squares, and $p_2 = (\frac{4}{5}, \frac{1}{10}, \frac{1}{10})$ plotted with triangles. For these three examples, the more the probability approaches the uniform distribution, the smaller the expected number of distinct factors.

The proof, presented in the rest of this section, consists in finding the frequency vector that contributes the most to the number of distinct α-gapped palindromic factors. This information will be crucial, as we will see that everything is concentrated on words with approximatively this frequency vector. We first work solely on the upper bound, then provide a matching lower bound.

3.1 Upper Bound for the Probability of a Given α-Gapped Pattern

In this section we compute an upper bound for the probability that a given α-gapped palindromic factor $uv\overline{u}$ appears at position j in a random word w of length n, with $i \in [n+1-|uvu|]$. We introduce several real variables to express this probability in a convenient way: Let ℓ be positive real such that $|u| = \ell \log n$. Let r be the non-negative real such that $|v| = r|u|$. The length of $uv\overline{u}$ is therefore $(2+r)\ell \log n$, and $r \leq \alpha - 1$ because of the α-gapped condition. Let also $\boldsymbol{x} = (x_1, \ldots, x_k)$ be the frequency vector of u and let $\boldsymbol{y} = (y_1, \ldots, y_k)$ be the frequency vector of v.

The probability that $uv\overline{u}$ appears as a factor at position j of a random word of length n is $p_n(\boldsymbol{x}, \boldsymbol{y}, \ell, r) := \mathbb{P}_n(uv\overline{u} \text{ factor at position } j)$ defined by

$$p_n(\boldsymbol{x}, \boldsymbol{y}, \ell, r) = \prod_{i=1}^{k} p_j^{2x_i \ell \log n + y_i r \ell \log n} = n^{\ell \sum_{i \in [k]} (2x_i + r y_i) \log p_i}. \tag{1}$$

In particular, this probability does not depend on the position j.

By linearity of the expectation, the expected number of occurrences of $uv\overline{u}$ in a random word of length n is simply $(n+1-|uv\overline{u}|)p_n(\mathbf{x}, \mathbf{y}, \ell, r)$. Since the probability that a word appears as a factor is bounded from above by its expected number of occurrences, the probability $q_n(\mathbf{x}, \mathbf{y}, \ell, r)$ that $uv\overline{u}$ is factor of a random word of length n satisfies

$$q_n(\mathbf{x}, \mathbf{y}, \ell, r) \leq n p_n(\boldsymbol{x}, \boldsymbol{y}, \ell, r) = n^{1+\ell \sum_{i \in [k]} (2x_i + r y_i) \log p_i}. \tag{2}$$

This upper bound can be greater than one, if the exponent is positive. Thus, we will use the following upper bound for the probability $q_n(\mathbf{x}, \mathbf{y}, \ell, r)$:

$$q_n(\mathbf{x}, \mathbf{y}, \ell, r) \leq \min(1, n p_n(\boldsymbol{x}, \boldsymbol{y}, \ell, r)) \leq n^{\min(0, 1+\ell \sum_{i \in [k]} (2x_i + r y_i) \log p_i)}. \tag{3}$$

This idea that the probability of appearance is bounded by the minimum between 1 and the expected number of occurrences is central in [3,11].

3.2 Upper Bound for Given Frequency Vectors

The result of this section is the following lemma.

Lemma 1. *Let* $\mathbf{x} = (x_1, \ldots, x_k)$ *and* $\mathbf{y} = (y_1, \ldots, y_k)$ *be two frequency vectors, such that* $(\ell \log n)\mathbf{x}$ *and* $(r\ell \log n)\mathbf{y}$ *are integer-valued. The expected number of distinct* α*-gapped factors of the form* $uv\overline{u}$ *in a random word of length* n*, such that* $|u| = \ell \log n$ *and* $|v| = r\ell \log n$*, and such that* u *has frequency vector* $\mathbf{x}$ *and*

v has frequency vector $\mathbf{y}$, is bounded from above by $\lambda n^{\ell \min(J_r(\mathbf{x},\mathbf{y}),K_{\ell,r}(\mathbf{x},\mathbf{y}))}$, for some positive constant λ, where $J_r(\mathbf{x},\mathbf{y})$ and $K_{\ell,r}(\mathbf{x},\mathbf{y})$ are defined by

$$J_r(\mathbf{x},\mathbf{y}) = H(\mathbf{x}) + rH(\mathbf{y}) \text{ and } K_{\ell,r}(\mathbf{x},\mathbf{y}) = J_r(\mathbf{x},\mathbf{y}) + \frac{1}{\ell} + \sum_{i=1}^{k}(2x_i + ry_i)\log p_i.$$

Proof. Let $\mu_n(\mathbf{x},\mathbf{y},\ell,r)$ be the expected number of distinct α-gapped factors described in the statement of the lemma. By linearity of the expectation, we have

$$\mu_n(\mathbf{x},\mathbf{y},\ell,r) = \sum_{\substack{u\in\mathcal{W}_{\ell\log n}(\mathbf{x})\\ v\in\mathcal{W}_{r\ell\log n}(\mathbf{y})}} q_n(\mathbf{x},\mathbf{y},\ell,r) = |\mathcal{W}_{\ell\log n}(\mathbf{x})|\cdot|\mathcal{W}_{r\ell\log n}(\mathbf{y})|\cdot q_n(\mathbf{x},\mathbf{y},\ell,r).$$

So we just have to bound from above the cardinalities of $\mathcal{W}_{\ell\log n}(\mathbf{x})$ and of $\mathcal{W}_{r\ell\log n}(\mathbf{y})$ to conclude using Eq. (3). This is done using Lemma 1 of [3], which states that $|\mathcal{W}_{\ell\log n}(\mathbf{x})| \le Cn^{\ell H(\mathbf{x})}$ for some positive constant C. □

Observe that there can be no possibilities for u or v if the values of $\mathbf{x}$, $\mathbf{y}$, ℓ, n and r are such that one of the quantities $x_i\ell\log n$ or $y_i\ell\log n$ is not an integer. This is not a problem, as our statement is an upper bound.

3.3 Optimizing $J_r(\mathbf{x},\mathbf{y})$ and $K_{\ell,r}(\mathbf{x},\mathbf{y})$, for Fixed ℓ and r

In this section we start to work on the upper bound provided by Lemma 1 by studying separately the two functions $J_r(\mathbf{x},\mathbf{y})$ and $K_{\ell,r}(\mathbf{x},\mathbf{y})$. The following lemma is a consequence of the properties of the entropy function.

Lemma 2. *For given $r \ge 0$, the function $J_r(\mathbf{x},\mathbf{y})$ is maximized on the set of probability vectors for $\mathbf{x}$ and $\mathbf{y}$ when $\mathbf{x} = \mathbf{y} = \overline{\mathbf{x}}$, where $\overline{\mathbf{x}} = (\frac{1}{k},\ldots,\frac{1}{k})$ is the uniform probability vector. For these values we have $J_r(\overline{\mathbf{x}},\overline{\mathbf{x}}) = (1+r)\log k$.*

The analysis of $K_{\ell,r}$ is not really complicated either. For any non-negative c, we define the probability vector $\mathbf{x}(c)$ by

$$\mathbf{x}(c) = \left(\frac{p_1^c}{\Phi(c)},\ldots,\frac{p_i^c}{\Phi(c)},\ldots,\frac{p_k^c}{\Phi(c)}\right).$$

In particular $\mathbf{x}(0)$ is the uniform probability vector $\overline{\mathbf{x}}$ and $\mathbf{x}(1) = \mathbf{p}$ is the distribution of the source. Recall also Gibbs' inequality [9], stating that if $\boldsymbol{s} = (s_1,\ldots,s_k)$ and $\boldsymbol{t} = (t_1,\ldots,t_k)$ are two probability vectors, then we have[2] $-\sum_{i=1}^k s_i\log s_i \le -\sum_{i=1}^k s_i\log t_i$, with equality if and only if $\boldsymbol{s} = \boldsymbol{t}$.

Lemma 3. *For given $r \ge 0$ and $\ell > 0$, the function $K_{\ell,r}(\mathbf{x},\mathbf{y})$ is maximized on the set of probabilities vector for $\mathbf{x}$ and $\mathbf{y}$ when $\mathbf{x} = \mathbf{x}(2)$ and $\mathbf{y} = \mathbf{p}$. For these values we have $K_{\ell,r}(\mathbf{x}(2),\mathbf{p}) = \frac{1}{\ell} + \log\Phi(2)$.*

[2] The case where some coordinates are zero is covered by setting $x\log x = 0$ for $x = 0$.

Proof. We can rewrite $K_{\ell,r}(\mathbf{x},\mathbf{y})$ the following way:

$$K_{\ell,r}(\mathbf{x},\mathbf{y}) = \left(H(\mathbf{x}) + \sum_{i=1}^{k} x_i \log p_i^2\right) + r\left(H(y) + \sum_{i=1}^{k} y_i \log p_i\right) + \frac{1}{\ell}.$$

The second parenthesis is non-positive by direct application of Gibbs' inequality, and maximal for $\mathbf{y} = \mathbf{p}$, in which case it is equal to zero. For the first parenthesis, Gibbs' inequality also applies as

$$H(\mathbf{x}) + \sum_{i=1}^{k} x_i \log p_i^2 = -\sum_{i=1}^{k} x_i \log x_i + \sum_{i=1}^{k} x_i \log \frac{p_i^2}{\Phi(2)} + \log \Phi(2).$$

It is therefore maximal for $\mathbf{x} = \mathbf{x}(2)$, where it is equal to $\log \Phi(2)$. □

3.4 Optimizing $G_{\ell,r}(\mathbf{x},\mathbf{y})$, for Fixed ℓ and r

Let $G_{\ell,r}(\mathbf{x},\mathbf{y}) = \min(J_r(\mathbf{x},\mathbf{y}), K_{\ell,r}(\mathbf{x},\mathbf{y}))$. This is the function to maximize in order to locate the maximum of the upper bound given in Lemma 1. We distinguish three cases (recall that J_ℓ is maximal at $(\overline{\mathbf{x}},\overline{\mathbf{x}})$ and $K_{\ell,r}$ at $(\mathbf{x}(2),\mathbf{p})$):

(a) If $J_r(\overline{\mathbf{x}},\overline{\mathbf{x}}) \leq K_{\ell,r}(\overline{\mathbf{x}},\overline{\mathbf{x}})$: then $G_{\ell,r}(\overline{\mathbf{x}},\overline{\mathbf{x}}) = J_r(\overline{\mathbf{x}},\overline{\mathbf{x}})$, and thus, $G_{\ell,r}$ is maximal at $(\overline{\mathbf{x}},\overline{\mathbf{x}})$, since $G_{\ell,r}(\mathbf{x},\mathbf{y}) \leq J_r(\mathbf{x},\mathbf{y})$.

(b) If $K_{\ell,r}(\mathbf{x}(2),\mathbf{p}) \leq J_r(\mathbf{x}(2),\mathbf{p})$: for the same reasons, $G_{\ell,r}$ is maximal at the point $(\mathbf{x}(2),\mathbf{p})$.

(c) If $J_r(\overline{\mathbf{x}},\overline{\mathbf{x}}) > K_{\ell,r}(\overline{\mathbf{x}},\overline{\mathbf{x}})$ and $K_{\ell,r}(\mathbf{x}(2),\mathbf{p}) > J_r(\mathbf{x}(2),\mathbf{p})$: since both J_r and $K_{\ell,r}$ are strictly concave (on the set of pairs of probabilities vectors), then $G_{\ell,r}$ has no local maximum on the set defined by $\{J_r(\mathbf{x},\mathbf{y}) < K_{\ell,r}(\mathbf{x},\mathbf{y})\}$, nor on the set $\{J_r(\mathbf{x},\mathbf{y}) > K_{\ell,r}(\mathbf{x},\mathbf{y})\}$. The global maximum of $G_{\ell,r}$ therefore lies on the set defined by the condition $J_r(\mathbf{x},\mathbf{y}) = K_{\ell,r}(\mathbf{x},\mathbf{y})$.

For convenience, we introduce the set $\mathcal{E}_{\ell,r}$ defined by $\mathcal{E}_{\ell,r} = \{(\mathbf{x},\mathbf{y}) : J_r(\mathbf{x},\mathbf{y}) = K_{\ell,r}(\mathbf{x},\mathbf{y})\}$.

It is not difficult to identify the ranges for the different cases depending on the values of ℓ and r. The properties of the first two cases are summarized in the lemma below.

Lemma 4. *The first two cases are characterized as follows:*

- *We are in Case (a) if and only if* $(2+r)\log k < \frac{1}{\ell}$. *In this case,* $G_{\ell,r}$ *has its maximum at* $(\overline{\mathbf{x}},\overline{\mathbf{x}})$, *with* $G_{\ell,r}(\overline{\mathbf{x}},\overline{\mathbf{x}}) = (2+r)\log k$.
- *We are in Case (b) if and only if* $H(\mathbf{x}(2)) + rH(p) + \log\Phi(2) > \frac{1}{\ell}$. *In this case,* $G_{\ell,r}$ *has its maximum at* $(\mathbf{x}(2),\mathbf{p})$, *with* $G_{\ell,r}(\mathbf{x}(2),\mathbf{p}) = \frac{1}{\ell} + \log\Phi(2)$.

Observe that each condition in Lemma 4 defines an open subset of the (ℓ, r) domain, each delimited by a hyperbole segment. These two subsets cannot intersect, since in each domain $G_{\ell,r}$ is maximized at a different point (recall that we assumed $\mathbf{p} \neq \overline{\mathbf{x}}$). Thus, the closed subset of the (ℓ, r) domain defining Case (c) cannot be empty. Our result for Case (c), which constitutes the main technical contribution of this article, is the following.

Lemma 5. *Let $\ell > 0$ and $r \geq 0$ be two real numbers such that $(2+r)\log k \geq \frac{1}{\ell}$ and $H(\mathbf{x}(2))+rH(p)+\log \Phi(2) \leq \frac{1}{\ell}$. Then $G_{\ell,r}(\mathbf{x},\mathbf{y})$ reaches its unique maximum for $\mathbf{x} = \mathbf{x}(2c)$ and $\mathbf{y} = \mathbf{x}(c)$, where $c \in (0,1)$ is the unique positive solution of the equation*

$$1 + \ell \frac{\Phi'(2t)}{\Phi(2t)} + \ell r \frac{\Phi'(t)}{\Phi(t)} = 0. \tag{4}$$

Proof. By Lemma 4, the hypothesis of the lemma implies that we are in Case (c). Hence, $G_{\ell,r}$ reaches its maximum, for probability vectors of $\mathcal{E}_{\ell,r}$; this is equivalent to $\frac{1}{\ell} + \sum_i (2x_i + r y_i)\log p_i = 0$, which is a linear condition on the $2k$ variables defining $(\mathbf{x},\mathbf{y})$.

On the considered domain we have $G_{\ell,r}(\mathbf{x},\mathbf{y}) = J_r(\mathbf{x},\mathbf{y}) = K_{\ell,r}(\mathbf{x},\mathbf{y})$, so we take $J_r(\mathbf{x},\mathbf{y})$ which is easier to study. Its gradient, as a function of $2k$ non-negative variables (and thus, not only for probability vectors) is the following:

$$\frac{\partial J_r}{\partial x_i} = -1 - \log x_i, \text{ and } \frac{\partial J_r}{\partial y_i} = -r - r\log y_i, \ \forall i \in [k].$$

Since we are looking for a pair of probability vectors $(\mathbf{x},\mathbf{y})$ that lies in the set $\mathcal{E}_{\ell,r}$ the following linear constraints must be satisfied: $\sum_{i\in[k]} x_i = 1$, $\sum_{i\in[k]} y_i = 1$ and $\frac{1}{\ell} + \sum_{i\in[k]}(2x_i + r y_i)\log p_i = 0$. Let $\boldsymbol{n}_1$, $\boldsymbol{n}_2$ and $\boldsymbol{n}$ be the three vectors normal to the constraints defined by

$$\boldsymbol{n}_1 = (\underbrace{1,\ldots,1}_{k \text{ times}},\underbrace{0,\ldots,0}_{k \text{ times}}); \quad \boldsymbol{n}_2 = (\underbrace{0,\ldots,0}_{k \text{ times}},\underbrace{1,\ldots,1}_{k \text{ times}});$$
$$\boldsymbol{n} = (2\log p_1,\ldots,2\log p_k, r\log p_1,\ldots,r\log p_k).$$

To locate our optimal value, we have to find where the gradient lies in the vector space spanned by $\boldsymbol{n}_1$, $\boldsymbol{n}_2$ and $\boldsymbol{n}$. We are thus looking for vectors $\mathbf{x}$ and $\mathbf{y}$ for which there exist three constants c_1, c_2, c_3 such that, for all $i \in [k]$,

$$-1 - \log x_i = c_1 + 2c_3 \log p_i, \text{ and } -r - r\log y_i = c_2 + c_3 r \log p_i.$$

By differences (keeping the equations involving x_1 and y_1), this leads to

$$\log\left(\frac{x_i}{x_1}\right) = -2c_3 \log\left(\frac{p_i}{p_1}\right) \text{ and } \log\left(\frac{y_i}{y_1}\right) = -c_3 \log\left(\frac{p_i}{p_1}\right).$$

Introducing parameter $c = -c_3$, we get that the x_i's must be proportional to p_i^{2c} (with a normalizing constant $\Phi(2c)$) and the y_i's must be proportional to p_i^c (with a normalizing constant $\Phi(c)$) for the *same* constant c. That is, we must take $(\mathbf{x},\mathbf{y}) = (\mathbf{x}(2c),\mathbf{x}(c))$ for some constant c; the parameters c_1 and c_2 are then easily recovered as, from the equation involving x_1:

$$c_1 = -1 - \log\left(\frac{p_1^{2c}}{\Phi(2c)}\right) + 2c\log p_i = -1 + \log \Phi(2c),$$

and, from the equation involving y_1, we have $c_2 = -r - r\log p_i + cr\log p_i$. Thus the only remaining equation is the one asking for $(\mathbf{x}, \mathbf{y})$ to be in the set $\mathcal{E}_{\ell,r}$. Setting $\mathbf{x} = \mathbf{x}(2c)$ and $\mathbf{y} = \mathbf{x}(c)$ in the equation, we get

$$-\frac{1}{\ell} = \frac{2}{\Phi(2c)} \sum_{i=1}^{k} p_i^{2c} \log p_i + \frac{r}{\Phi(c)} \sum_{i=1}^{k} p_i^{c} \log p_i, \tag{5}$$

or equivalently,

$$-\frac{1}{\ell} = 2\frac{\Phi'(2c)}{\Phi(2c)} + r\frac{\Phi'(c)}{\Phi(c)}. \tag{6}$$

Now consider the function $\frac{\Phi'}{\Phi}$ appearing on the right-hand side of (6). Its derivative is $\frac{\Phi''\Phi - \Phi'^2}{\Phi^2}$, which is strictly positive by an application of Cauchy-Schwarz inequality (we use here that $\mathbf{p} \neq \overline{\mathbf{x}}$). Thus, the right-hand side of (4) is an increasing, continuous function of c when r and ℓ are considered as fixed parameters. Evaluating this function for $c = 0$ yields a value less than $-1/\ell$, from the condition that we are not in Case (a), and for $c = 1$, a value more than $-1/\ell$, from the condition that we are not in Case (b). This proves that the equation has a unique solution c, and that this solution lies strictly between 0 and 1. □

3.5 Optimizing the Exponent on ℓ and r

At this point, we know where the maximum is when ℓ and r are fixed, for all three ranges. We still have to find the values of ℓ and r that maximize the exponent $\ell G_{\ell,r}(\mathbf{x}, \mathbf{y})$, in each case, for the maximal $\mathbf{x}$ and $\mathbf{y}$. Recall that $r \in [0, \alpha - 1]$ in our settings. The next two lemmas are directly derived from Lemma 4.

Lemma 6. *For Case (a), the maximum of $\ell G_{\ell,r}(\mathbf{x}, \mathbf{y})$ is reached for $\ell = \ell_0$, $r = \alpha - 1$ and $\mathbf{x} = \mathbf{y} = \overline{\mathbf{x}}$, where $\ell_0 = \frac{1}{(2+r)\log k}$; we have $\ell_0 G_{\ell_0,\alpha-1}(\overline{\mathbf{x}}, \overline{\mathbf{x}}) = \frac{\alpha}{\alpha+1}$.*

Lemma 7. *For Case (b), the maximum of $\ell G_{\ell,r}(\mathbf{x}, \mathbf{y})$ is reached for $\ell = \ell_1$, $r = \alpha - 1$, $\mathbf{x} = \mathbf{x}(2)$ and $\mathbf{y} = \mathbf{p}$, where $\ell_1 = \frac{1}{H(\mathbf{x}(2)) + rH(p) - \log \Phi(2)}$; we have $\ell_1 G_{\ell_1,\alpha-1}(\overline{\mathbf{x}}, \overline{\mathbf{x}}) = \frac{H(\mathbf{x}(2)) + (\alpha-1)H(p)}{H(\mathbf{x}(2)) + (\alpha-1)H(p) - \log \Phi(2)}$.*

Observe that ℓ_0 and ℓ_1 are not formally in the ranges for Case (a) and Case (b) as they have been defined. We allow this abuse of notation, as the exponent function can be extended by continuity at ℓ_0 and at ℓ_1.

We now focus on Case (c). By Lemma 5 we know that the maximum of $G_{\ell,r}$ is reached at some point $(\mathbf{x}(2c), \mathbf{x}(c))$ of $\mathcal{E}_{\ell,r}$, for some $c \in (0, 1)$ that is implicitly defined. Our optimization is on the variables ℓ and r, and c is viewed as a function of these two variables. We introduce the notation $h(c) = H(\mathbf{x}(c))$, for which an elementary calculation yields $h(c) = \log \Phi(c) - c\frac{\Phi'(c)}{\Phi(c)}$. As $G_{\ell,r} = J_r$ on $\mathcal{E}_{\ell,r}$, our problem reduces to maximizing the exponent $E(r, \ell, c) = \ell h(2c) + \ell r h(c)$ subject to the constraint $1 + 2\ell\frac{\Phi'(2c)}{\Phi(2c)} + \ell r\frac{\Phi'(c)}{\Phi(c)} = 0$. The solution is given in the following lemma.

Lemma 8. *For Case (c), the maximum of $\ell G_{\ell,r}(\mathbf{x},\mathbf{y})$ is reached for $\ell = \ell_{c^*}$, $r = \alpha - 1$, $\mathbf{x} = \mathbf{x}(2c^*)$ and $\mathbf{y} = \mathbf{x}(c^*)$, where c^* is the solution of the equation*

$$\Phi(2c)\Phi(c)^{\alpha-1} = 1. \tag{7}$$

At this point, the value of the exponent is $\ell_{c^*} G_{\ell_{c^*},\alpha-1}(\mathbf{x}(2c^*),\mathbf{x}(c^*)) = c^*$.

Proof. Remark that, using the simplification of $h(c)$, we have $E(\ell,r,c) = c + \ell\,(\log\Phi(2c) + r\log\Phi(c))$. Lets us consider c as an implicit function of (ℓ,r), so that E becomes a function of only two variables ℓ and r, for which we compute partial derivatives:

$$\begin{aligned}
\frac{\partial E}{\partial r}(\ell,r) &= \frac{\partial c}{\partial r}(\ell,r) + 2\ell\frac{\partial c}{\partial r}(\ell,r)\frac{\Phi'(2c)}{\Phi(2c)} + \ell\log\Phi(c) + \ell r\frac{\partial c}{\partial r}(\ell,r)\frac{\Phi'(c)}{\Phi(c)} \\
&= \ell\log\Phi(c) + \frac{\partial c}{\partial r}(\ell,r)\left(1 + 2\ell\frac{\Phi'(2c)}{\Phi(2c)} + \ell r\frac{\Phi'(c)}{\Phi(c)}\right) = \ell\log\Phi(c); \\
\frac{\partial E}{\partial \ell}(\ell,r) &= \frac{\partial c}{\partial \ell}(\ell,r) + \log\Phi(2c)) + r\ln(\Phi(c)) + \ell\left(2\frac{c}{\ell}\frac{\Phi'(2c)}{\Phi(2c)} + \frac{\partial c}{\partial \ell}(\ell,r)r\frac{\Phi'(c)}{\Phi(c)}\right) \\
&= \log\Phi(2c) + r\log\Phi(c).
\end{aligned}$$

The expression for $\frac{\partial E}{\partial r}$ shows it to be positive ($c < 1$ implies $\Phi(c) > 1$), so the maximum is obtained when r is as large as possible, namely, $r = \alpha - 1$. The expression for $\frac{\partial E}{\partial \ell}$ provides a candidate for a maximum, where it reaches zero. This happens for c solution of the equation

$$\log\Phi(2c) + (\alpha - 1)\log\Phi(c) = 0. \tag{8}$$

We compute the second derivative in ℓ to verify that it is a local maximum:

$$\begin{aligned}
\frac{\partial^2 E}{\partial \ell^2}(\ell,r) &= 2\frac{\partial c}{\partial \ell}(\ell,r)\frac{\Phi'(2c)}{\Phi(2c)} + r\frac{\partial c}{\partial \ell}(\ell,r)\frac{\Phi'(c)}{\Phi(c)} \\
&= \frac{\partial c}{\partial \ell}(\ell,r)\underbrace{\left(2\frac{\Phi'(2c)}{\Phi(2c)} + r\frac{\Phi'(c)}{\Phi(c)}\right)}_{=-\frac{1}{\ell}} = -\frac{1}{\ell}\frac{\partial c}{\partial \ell}(\ell,r).
\end{aligned}$$

A closer look at Eq. (4), the implicit equation for c, yields that c is an increasing function of ℓ. Hence $\frac{\partial^2 E}{\partial \ell^2}(\ell,r)$ is negative, and thus it has a unique maximum for c solution of Eq. (8). This equation is equivalent to Eq. (7), concluding the proof. □

Lemmas 6, 7 and 8 describe the various maximum exponents obtained in the three cases; we now combine the three into a single result. It is obtained by remarking that in all three cases, the maximum exponent is reached on the line $r = \alpha - 1$, and that it is a continuous function of ℓ.

Proposition 1. *The maximum exponent over all choices of ℓ, r, $\mathbf{x}$ and $\mathbf{y}$ is obtained for Case (c), namely, $r = \alpha - 1$, $\ell = \ell_{c^*}$ where $\Phi(2c^*)\Phi(c^*)^{\alpha-1} = 1$, $\mathbf{x} = \mathbf{x}(2c^*)$ and $\mathbf{y} = \mathbf{x}(c^*)$; this maximum exponent is equal to c^*.*

3.6 Proof of Theorem 1

Up to now, we have optimized over continuous domains for ℓ, r, $\mathbf{x}$ and $\mathbf{y}$. We now deal properly with the fact that they must be rational numbers and vectors that correspond to actual composition vectors for words. We let ℓ^*, c^*, $\mathbf{x}^* = (x_1^*, \ldots, x_k^*)$ and $\mathbf{y}^* = (y_1^*, \ldots, y_k^*)$ denote the real-valued optimal solution describe in the previous section. Let $L = \lfloor \ell^* \log n \rfloor$, $R = \lfloor (\alpha - 1)L \rfloor$, and define the vectors $\mathbf{x}$ and $\mathbf{y}$ as follows: for any $i \in [k-1]$, $x_i = \frac{1}{L}\lfloor x_i^* L \rfloor$ and $y_i = \frac{1}{R}\lfloor y_i^* R \rfloor$; finally, let $x_k = 1 - \sum_{i<k} x_i$ and let $y_k = 1 - \sum_{i<k} y_i$. Defined this way, $L\mathbf{x}$ and $R\mathbf{y}$ are vectors of integers summing to L and R respectively, so we can look at the expected number of distinct α-gapped palindromic factors $uv\overline{u}$ where u has composition exactly $L\mathbf{x}$, and v has composition exactly $L\mathbf{y}$; we write this expected number as n^c (equivalently, c is the logarithm to base n of the expected number of factors).

Since $||\mathbf{x} - \mathbf{x}^*|| = O(1/\log n)$ and $||\mathbf{y} - \mathbf{y}^*|| = O(1/\log n)$, we have that $c = c^* - O(1/\log n)$ (this would be $1/\log^2 n$ if both partial derivatives of E vanished at $(\ell^*, \alpha - 1)$, but $\partial E/\partial r$ does not vanish). Thus, we have $n^c = n^{c^* - O(\log n)} = e^{c^* \log n - O(1)} = \Theta(n^{c^*})$.

Now let $\ell^+ = \frac{3}{|\log p_{\max}|}$. For any choice of ℓ and r with $\ell^+ \le \ell \le n/\log n$ and $0 \le R \le \alpha \ell \log n$, the probability of having an α-gapped palindromic factor $uv\overline{u}$ at some position j, with $|u| = \ell \log n$ and $|v| = R$, is at most $p_{\max}^{\ell \log n} \le n^{-3}$: for any choice of u and v, the probability that the next $\ell \log n$ letters are exactly those of u is upper bounded by $p_{\max}^{\ell \log n}$. Since there are fewer than n^3 choices for the triple $(|u|, |v|, j)$, the expected number of such "long" factors is less than 1.

As a consequence, the dominant contribution to the expected number of gapped palindromic factors comes from those with $|u| \le \ell^+ \log n$. Each possible composition vector for u and v contributes less than n^{c^*}, and there are at most $(\ell^+ \log n)^{2k}(\alpha - 1)^k = \tilde{O}(1)$ such composition vectors; thus the $\tilde{\Theta}(n^{c^*})$ bound carries over for the expected total number of distinct α-gapped palindromic factors of all possible lengths. □

4 Typical Composition Vectors of Palindromic Factors

In this section, we show that with asymptotic probability 1, *most* gapped palindromic factors present in a large random word will be as described in the upper bound computations of the previous section. For this, we must first prove that our previous result on the *expected* number of gapped palindromic factors hold with good enough probability for the random variable that counts these factors.

Theorem 2. *There exist two constants $a < 0$ and $b > 0$ such that, with asymptotic probability 1 (when n tends to infinity), the number $\Gamma_{\alpha,n}$ of distinct α-gapped palindromic factors in a random word of length n lies between $n^{c^*} \log^a(n)$ and $n^{c^*} \log^b(n)$.*

Proof. The upper bound is a direct consequence of applying Markov's inequality to the bound on the expectation of Theorem 1: if we simply multiply by $\log^\varepsilon n$

the upper bound in the $\tilde{\Theta}(n^{c^*})$ in the theorem, the probability that $\Gamma_{\alpha,n}$ is higher than this new bound is $O((\log n)^{-\varepsilon}) = o(1)$.

For the lower bound, we now prove that with high enough probability, $\Gamma_{\alpha,n}$ is at least $\mathbb{E}(\Gamma_{\alpha,n})/\log n^d$ for some $d > 0$. We will do this by proving such a lower bound for the factors appearing in a subset of the possible positions in the word: Let $m = n/(2+\alpha)\ell^* \log n$. Our word of length n is obtained[3] by concatenating m independent words $W_1, \ldots, W_m$, each of length $(2+\alpha)\ell^* \log n$.

Now, for each possible α-gapped palindrome w of length $(2+\alpha)\ell^* \log n$, and for each integer $1 \leq i \leq m$, define the Bernoulli random variable $X_{i,w}$ as 1 if $W_i = w$, and 0 otherwise; then define X_w as $\max_i X_{i,w}$, *i.e.*, $X_w = 1$ if and only if w appears in a factor in one of the m positions in the whole random word. Finally, set $X = \sum_w X_w$: X is the total number of distinct α-gapped palindromic factors that appear in at least one of the m positions. Thus, $X \leq G_{\alpha,n}$, but $\mathbb{E}(X) = \tilde{\Theta}(n^{c^*})$ (we lose a factor of at most $\log n$ because we only consider $\Theta(n/\log n)$ positions instead of n, but this is absorbed by the $\tilde{\Theta}$ notation).

The collection of random variables $(X_{i,w})$ is negatively associated in the sense of [2], so that (by [2], Proposition 7), the classical Chernoff-Hoeffding bounds apply to X. This is enough to prove (using, for example, [10], Theorem 4.2) that the probability of X being less than half its expectation is exponentially small. □

Our final result ensures that, with probability close to 1, almost all α-gapped palindromic factors that appear in the random word have composition vectors close to the typical vectors described earlier.

Theorem 3. *We again let $\Gamma_{\alpha,n}$ denote the total number of distinct α-gapped palindromic factors of a random word of length n; and, for any $\varepsilon > 0$, we let $\Gamma_{\alpha,n,\varepsilon}$ denote the total number of these factors whose frequency vectors lie within distance ε of the optimal vectors $\mathbf{x}^*$ and $\mathbf{y}^*$. Then, for any $\delta > 0$, with asymptotic probability 1, we have $\Gamma_{\alpha,n,\varepsilon} \geq (1-\delta)\Gamma_{\alpha,n}$.*

Proof. We already know that, with asymptotic probability 1, $\Gamma_{\alpha,n,\varepsilon} \geq n^{c^*}/\log n^a$ for some $a > 0$. From the proof of Lemma 8, we know that any frequency vectors at distance at least ε from $(\mathbf{x}, \mathbf{y})$ come with an exponent at most $c^* - \beta\varepsilon^2$ (for some $\beta > 0$ which only depends on the second derivative of E at the critical point), so that the total expected number of distinct α-gapped factors with such frequency vectors is $\tilde{\Theta}(n^{c^*-\beta\varepsilon^2})$. Markov's inequality turns this into a high probability bound at the cost of logarithmic factors absorbed into the $\tilde{\Theta}$ notation, and with high probability, $\Gamma_{\alpha,n,\varepsilon}$ is within a factor $1 - \tilde{O}(n^{-\beta\varepsilon^2})$ of $\Gamma_{\alpha,n}$.

5 Conclusion

In this article we show that the expected number of distinct α-gapped palindromic factors in a random word of length n is $\tilde{\Theta}(n^{c^*})$, where c^* is implicitly

[3] We disregard rounding errors in lengths; properly dealing with them by means of integer parts would only yield clumsier notations without changing the asymptotic results.

defined as the solution of some equation depending on the probability $\mathbf{p}$ of the source and of α. Moreover, for any positive ε, the frequency vectors of u and v of such factors $uv\overline{u}$ are likely to be at distance at most ε of $\mathbf{x}^*$ and $\mathbf{y}^*$.

To conclude, we want to emphasize that the techniques we used follow those we introduced in [3]. These methods therefore prove useful to study the number of distinct elements (palindromes, subwords, ...) in different settings, for random words generated by a memoryless distribution.

References

1. Crochemore, M., Kolpakov, R., Kucherov, G.: Optimal bounds for computing α-gapped repeats. In: Dediu, A.-H., Janoušek, J., Martín-Vide, C., Truthe, B. (eds.) LATA 2016. LNCS, vol. 9618, pp. 245–255. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30000-9_19
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.: On the biased partial word collector problem. In: Bender, M.A., Farach-Colton, M., Mosteiro, M.A. (eds.) LATIN 2018. LNCS, vol. 10807, pp. 413–426. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77404-6_30
4. 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, Warsaw, Poland, 4–6 July 2017. LIPIcs, vol. 78, pp. 21:1–21:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017)
5. Dumitran, M., Gawrychowski, P., Manea, F.: Longest gapped repeats and palindromes. Discrete Math. Theoret. Comput. Sci. **19**(4) (2017)
6. Gawrychowski, P., I, T., Inenaga, S., Köppl, D., Manea, F.: Tighter bounds and optimal algorithms for all maximal α-gapped repeats and palindromes - finding all maximal α-gapped repeats and palindromes in optimal worst case time on integer alphabets. Theory Comput. Syst. **62**(1), 162–191 (2018)
7. Kolpakov, R., Kucherov, G.: Searching for gapped palindromes. Theoret. Comput. Sci. **410**(51), 5365–5373 (2009)
8. Kolpakov, R., Podolskiy, M., Posypkin, M., Khrapov, N.: Searching of gapped repeats and subrepetitions in a word. J. Discrete Algorithms **46-47**, 1–15 (2017)
9. MacKay, D.J.: Information Theory, Inference and Learning Algorithms. Cambridge University Press, Cambridge (2003)
10. Motwani, R., Raghavan, P.: Randomized algorithms. ACM Comput. Surv. (CSUR) **28**(1), 33–37 (1996)
11. Rubinchik, M., Shur, A.M.: The number of distinct subpalindromes in random words. Fundam. Inform. **145**(3), 371–384 (2016)
12. Rubinchik, M., Shur, A.M.: EERTREE: an efficient data structure for processing palindromes in strings. Eur. J. Comb. **68**, 249–265 (2018)
13. Tanimura, Y., Fujishige, Y., I, T., Inenaga, S., Bannai, H., Takeda, M.: A faster algorithm for computing maximal α-gapped repeats in a string. In: Iliopoulos, C.S., Puglisi, S.J., Yilmaz, E. (eds.) SPIRE 2015. LNCS, vol. 9309, pp. 124–136. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23826-5_13

Computational Complexity of Robot Arm Simulation Problems

Tianfeng Feng[1(✉)], Takashi Horiyama[2], Yoshio Okamoto[3], Yota Otachi[4], Toshiki Saitoh[5], Takeaki Uno[6], and Ryuhei Uehara[1]

[1] Japan Advanced Institute of Science and Technology (JAIST), Nomi, Japan
{ftflluy,uehara}@jaist.ac.jp
[2] Saitama University, Saitama, Japan
horiyama@al.ics.saitama-u.ac.jp
[3] RIKEN Center for Advanced Intelligence Project, University of Electro-Communications, Chofu, Japan
okamotoy@uec.ac.jp
[4] Kumamoto University, Kumamoto, Japan
otachi@cs.kumamoto-u.ac.jp
[5] Kyushu Institute of Technology, Kitakyushu, Japan
toshikis@ces.kyutech.ac.jp
[6] National Institute of Informatics, Tokyo, Japan
uno@nii.jp

Abstract. We consider a simulation problem of a general mechanism by a robot arm. A robot arm can be modeled by a path P, and the target is modeled by a general graph G. Then the problem asks if there is an edge-weighted Eulerian path of G spanned by P. We first show that it is strongly NP-hard even if edge lengths are restricted. Then we consider two different variants of this problem. We first allow the edges in P to be elastic, and minimize the elastic ratio when G is a path. Second, we allow P to cover an edge of G twice or more. The problem is weakly NP-hard even if G is an edge. We thus assume that each edge of G is covered by P exactly twice, and obtain three hardness results and one polynomial-time algorithm when G and edge lengths are restricted.

Keywords: Edge-weighted Eulerian path problem
Graph spanning problem · Linkage · Robot arm

1 Introduction

A *robot arm* is a type of programmable mechanical arms, which can be modeled by a *linkage*. A linkage is a collection of fixed-length 1D segments joined at their endpoints forming a path. (See [1] for further details.) Namely, a linkage is a path $P = (v_1, v_2, \ldots, v_n)$ with length function $\ell : E \to \mathbb{R}$, where v_i is an endpoint, $e_i = \{v_i, v_{i+1}\}$ is an edge in $E = \{\{v_i, v_{i+1}\} \mid 1 \leq i \leq n-1\}$, and its length is given by $\ell(e_i)$. Now we consider the following situation (Fig. 1). You are given a general target mechanism which is modeled by a graph $G = (V', E')$,

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 177–188, 2018.
https://doi.org/10.1007/978-3-319-94667-2_15

and a robot arm modeled by a linkage $P = (V, E)$ as above with length function $\ell : E \cup E' \to \mathbb{R}$. Our mission is *to simulate* the target graph G by the given linkage P. The joints in P are programmable, and each joint (or vertex) of G should be simulated by a joint of P, however we can also put the joints of P on some internal points of edges of G because they can be fixed. Therefore, our problem can be formalized as finding the following mapping ϕ from P to G:

- Each vertex of G should be mapped from some vertices of P;
- Each edge of G should be mapped from a subpath of P by ϕ;
- Each edge of P should be mapped to on an edge of G, which may span from one internal point to another on the edge.

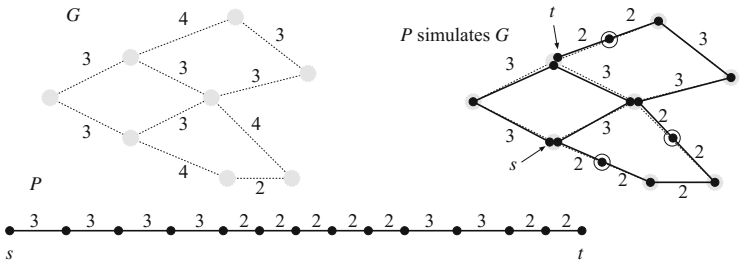


Fig. 1. A simple example. A robot arm modeled by P can simulate a given mechanism modeled by the graph G as shown in the figure. When P simulates G, each circled joint is fixed, and two joints of the robot arm on the same vertex of G move with synchronization.

The decision problem asks if there exists a mapping ϕ from P to G. That is, it asks if there is an Eulerian path of G spanned by P such that (1) when P visits a vertex in G, a vertex of P should be put on it, and (2) some vertices in P can be put on internal points of edges of G. When all edges in P and G have the same length, it is easy to solve that in linear time since the problem is the ordinary Eulerian path problem. In the context of formal languages, there are some variants of the Eulerian path problem with some constraints (see [2] for a comprehensive survey). However, as far as the authors know, the *robot arm simulation problem*, our variant of the Eulerian path problem has not been investigated, while the situation is quite natural.

The first interesting result is that this problem is strongly NP-hard even if edge lengths are quite restricted. Precisely, the problem is strongly NP-hard even if P and G consist of edges of lengths only 1 and 2 (Theorem 1). We remind that if they consist of unit length edges, the problem is linear time solvable. We thus tackle this problem in two different ways.

The first problem is an optimization version of this problem. In this variant, we consider a linkage is *elastic*, that is, the length of one line segment is not fixed and can be changed a little bit. This situation is natural not only in the

context of the robot arm simulation, but also in the approximation algorithm. Formally, we allow the edges in P to be elastic to fit the vertices of P to ones of G. Our goal is to minimize the stretch/shrink ratio of each edge of P. We show that when G is a path, this can be solved in polynomial time by dynamic programming.

In the second way, we allow P to cover an edge of G twice or more. In this situation, we can simulate G by P even if G does not have an Eulerian path. In this case, we do not allow P to be elastic, or its ratio is fixed to 1. We first show that the problem is weakly NP-hard even if G is an edge (Theorem 3). In fact, this problem is similar to the ruler folding problem (see, e.g., [1,3]). From the viewpoint of a simulation of G by P, we can take the following strategy in general. For a given G, first make a spanning tree T of G, and traverse T in the depth first manner. It is easy to see that this strategy works for any connected graph G even if G does not have an Eulerian path, and the trail of the traverse of T gives us a way of simulation of G by P in a sense. In this paper, we focus on the case that the graph G is a tree. Precisely, we consider the following problem: For a given tree G and a path P (with edge lengths), the traverse problem asks if G has a trail by P such that each edge of G is traversed exactly twice. (We note that trees form a representative class of graphs that have no Eulerian paths.) We first mention that this problem is quite easy when each edge has unit length. The answer is yes if and only if $G = (V, E)$ is connected and P contains $2|E|$ edges. From the practical viewpoint, it seems to be reasonable when we simulate a mechanism by a robot arm. However, this problem is still strongly NP-hard even in quite restricted cases; (1) G is a star, and P consists of edges of only two different lengths, and (2) G is a spider, and all edges are of two different lengths. On the other hand, the problem is polynomial time solvable when G is a star and its edge lengths are of k different values.

2 Preliminaries

In this paper, we only consider a simple undirected graph $G = (V, E)$. A *path* $P = (v_1, v_2, \ldots, v_n)$ consists of n vertices with $n-1$ edges e_i joining v_i and v_{i+1} for each $i = 1, \ldots, n-1$. The vertices v_1 and v_n of the path are called *endpoints*. Let $K_{n,m}$ denote a *complete bipartite graph* $G = (X, Y, E)$ such that $|X| = n$, $|Y| = m$, and every pair of a vertex in X and a vertex in Y is joined by an edge. A graph $G = (V, E)$ is a *tree* if it is connected and acyclic. Here a graph G is a *star* if and only if it is a complete bipartite graph $K_{1,n-1}$, and a graph G is a *spider* if and only if G is a tree that has only one vertex of degree greater than 2. In a star or a spider, the unique vertex of degree greater than or equal to 3 is called *center*. Without loss of generality, we assume that a star or a spider always has the center. Let $G = (V', E')$ and $P = (V, E)$ be a graph and a path $(v_1, v_2, \ldots, v_n)$. Let $\ell : E' \cup E \to \mathbb{R}$ be an edge-length function of them. We say the linkage P can *simulate* the mechanism G if each edge in G is spanned by at least one subpath of P, and no subpath of P properly joins two non-adjacent vertices in G. We formalize the notion of simulation by a mapping ϕ that maps

each vertex V in P to a *point* in G as follows. For any edge $e = \{u, v\} \in E'$, we consider e a line segment (u, v) of length $\ell(e)$. Then the *intermediate point* p at distance $t\ell(e)$ from u is denoted by $p = tv + (1-t)u$, where $0 < t < 1$. We note that the endpoints of an edge e are not considered intermediate points of e. Now we first define a set of *points* in G by V' and all intermediate points on edges of E'. Then we define a mapping ϕ from V to points of G as follows. To make it clear, we first divide V in P into two subsets V_e and V_i such that each vertex in V_e is mapped to a vertex in V', and each vertex in V_i is mapped to an intermediate point of G. In our problems, we assume that $\phi(v_0)$ and $\phi(v_n)$ should be in V_e. That is, P should start and end at vertices in G. Depending on the restrictions, we consider some different simulation problems as follows.

Weighted Eulerian path problem: We consider the mapping ϕ from V_e to V' that satisfies some conditions as follows; (1) for every $v' \in V'$, there is at least one vertex $v \in V_e$ with $\phi(v) = v'$; (2) for each edge $e' = \{v', u'\} \in E'$, there is a pair of vertices v_i and v_j in V_e such that (2a) $\ell(e') = \sum_{k=i}^{j-1} \ell(e_k)$, and (2b) there is no other vertex v_k is in V_e between v_i and v_j. Intuitively, each edge e' in G corresponds to a subpath in P, and vice versa. In other words, some vertices in P are mapped to some intermediate points in G, and the corresponding joints of the robot arm are fixed when the robot arm P simulates the target mechanism G. We note that by the length condition (2a), we can assume that when the subpath $(v_i, \ldots, v_j)$ in P simulates an edge $e = \{\phi(v_i), \phi(v_j)\}$, it is not allowed to span it in a zig-zag way. We also add one condition: (3) for each edge (v_k, v_{k+1}) in P, $\{\phi(v_k), \phi(v_{k+1})\}$ should be on an edge e in G. That is, there is a subpath $(v_i, \ldots, v_k, v_{k+1}, \ldots, v_j)$ with $i \leq k < k+1 \leq j$ such that (3a) $e = \{\phi(v_i), \phi(v_j)\}$ and $\phi(v_k)$ is on an intermediate point on e for each $i < k < j$. In other words, all edges in P are used for spanning some edges in G, and there is no other subpath in P joining two endpoints in G by the length condition (2a). Then we say that the linkage P can *simulate* the mechanism G if there is a mapping ϕ satisfying the conditions (1), (2), and (3). This problem can be seen as the Eulerian path problem with edge weights.

Elastic linkage problem: In this problem, we allow all edges in P to be elastic to simulate the path G by the path P. The *elastic ratio* of an edge e is defined by $\max\{l'/l, l/l'\}$, where l is the length of the edge $e = \{u, v\}$ in P and l' is the length of the edge $\{\phi(u), \phi(v)\}$ in G. (Intuitively, the length of edge e is changed from l on $P$to l' on G.) For a given graph $G = (V', E')$ and a path $P = (V, E)$, it is easy to observe that P can simulate G (with elastic edges) if and only if G has an Eulerian path and $|V'| \leq |V|$. When G has an Eulerian path by P with elastic edges, the *elastic ratio* of the mapping is defined by the maximum elastic ratio of all edges in P. Then the elastic linkage problem asks to minimize the elastic ratio of the mapping from P to G for given G and P.

Traverse problem by a path: In this variant, we allow the mapping to map two subpaths of P to an edge of G, however, we do not allow P to be elastic, or its ratio is fixed to 1. For a given graph $G = (V', E')$ and a path $P = (V, E)$,

when all edges have a unit length, it is easy to observe that P can simulate G in this manner if and only if G is connected and $2|E| = |E'|$. We first perform the depth first search on G, and traverse this search tree. We also consider its edge-weighted version as the *traverse problem* for a graph G and a path P. (In this paper, in fact, we only investigate the cases that G is a tree.)

In this paper, we will often use the following problem to show the hardness of our problems.

3-Partition Problem

Input: An integer B and a multiset A of $3m$ integers $A = \{a_1, a_2, \ldots, a_{3m}\}$ with $B/4 < a_i < B/2$.

Output: Determine if A can be partitioned into m multisets $S_1, S_2, \ldots, S_m$ such that $\sum_{a_j \in S_i} a_j = B$ for every i.

Without loss of generality, we can assume that $\sum_{a_i \in A} a_i = mB$, and $|S_i| = 3$. It is well known that the 3-Partition problem is strongly NP-complete [1].

3 Weighted Eulerian Path Problem

Now we show the main theorem in this section.

Theorem 1. *Let P, G, ℓ be a path, an undirected graph, and a length function, respectively. Then the weighted Eulerian path problem is strongly NP-hard even if $\ell(e)$ is either 1 or 2 for any e in P and G.*

Proof. It is easy to see that the problem is in NP. Therefore we show the hardness. We reduce the 3-Partition problem to the weighted Eulerian path problem.

Let P_{B+1} be a path that consists of B consecutive edges of length 2, and P_4 be a path that consists of 3 consecutive edges of length 1. Then the path P is obtained by joining m subpaths P_{B+1} and m subpaths P_4 alternatingly, that is, P is constructed by joining $P_{B+1}, P_4, P_{B+1}, P_4, \ldots, P_4, P_{B+1}$, and P_4. The graph G is constructed as follows. For each i with $1 \leq i \leq 3m$, we construct a cycle C_{a_i} of a_i edges of length 2. We also construct m cycles C_3 of 3 edges of length 1. Then these $4m$ cycles share a special vertex c in common. That is, G is a cactus that consists of $4m$ cycles, and all vertices have degree 2 except the common vertex c that has degree $8m$. The construction is illustrated in Fig. 2.

It is easy to see that this is polynomial-time reduction. Thus, we show that A has a solution if and only if P can simulate G. We first observe that no edge of length 2 in P_{B+1} in P can cover a cycle C_3 in G. Therefore, when P covers G, every C_3 of G has to be covered by P_4 in P. Thus, each endpoint of P_4 should be on c in G, and no edge in P_{B+1} can cover edges in C_3. Hence, each subpath P_{B+1} in P covers exactly B edges in the set of cycles C_{a_i} that consists of edges of length 2. Since $B/4 < a_i < B/2$ for each i, each subpath P_{B+1} covers exactly three cycles C_{a_i}, C_{a_j} and C_{a_k} for some i, j, k with $a_i + a_j + a_k = B$. Clearly, each cover for a subpath P_{B+1} gives a subset of A, and the collection of these subsets gives us a solution to the 3-Partition problem and vice versa. □

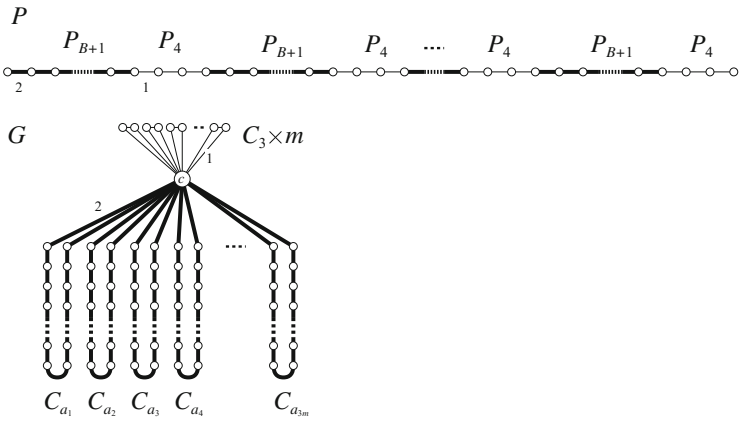


Fig. 2. Construction of P and G; bold lines are of length 2, and thin lines are of length 1. Each P_{i+1} consists of i edges and each C_i consists of i edges.

4 Elastic Linkage Problem

In this section, we consider the elastic linkage problem for two paths G and P:

Elastic linkage problem from path to path
Input: Two paths $G = (V', E')$ and $P = (V, E)$ with length function ℓ.
Output: a mapping ϕ with minimum elastic (or stretch/shrink) ratio.

In this problem, we allow all edges in P to be elastic to simulate the path G by the path P. We let G is a path $(u_1, u_2, \ldots, u_{n'})$ and P is a path $(v_1, v_2, \ldots, v_n)$. Without loss of generality, we assume that $n' \leq n$. Since each vertex in G should be mapped from only one vertex in P, it should be $\phi(v_1) = u_1$ and $\phi(v_n) = u_{n'}$, otherwise the elastic ratio will be infinity. We show a polynomial-time algorithm for this problem based on a dynamic programming.

First, we show a technical lemma when G is just an edge. In this case, the optimal value is achieved when all ratios are even.

Lemma 1. *Assume that G consists of an edge $e = (u_1, u_2)$. When $P = (V, E)$ is a path, the minimum elastic ratio is achieved when the ratio of each $e \in E$ takes the same value.*

Proof. Assume that the length of the edge $e = (u_1, u_2)$ in G is L, $E = \{e_1, e_2, \ldots, e_{n-1}\}$, and the length of each e_i is l_i. For a mapping ϕ, let r_i be the ratio of the edge e_i for each $i = 1, 2, \ldots, n-1$. Then we have $r_1 l_1 + r_2 l_2 + \cdots + r_{n-1} l_{n-1} = L$.

Assume the maximum among r_i for all $1 \leq i \leq n-1$ is r_k, and the minimum among r_i for all $1 \leq i \leq n-1$ is r_h. Thus, it is obvious that $r_k \geq L/(l_1 + l_2 + \cdots + l_{n-1})$, $1/r_h \geq (l_1 + l_2 + \cdots + l_{n-1})/L$.

According to the definition, the elastic ratio er of this mapping is the maximum among r_i and its reciprocal for all $1 \leq i \leq n-1$. That is, er equals the larger of r_k and $1/r_h$.

When $r_1 = r_2 = \cdots = r_{n-1}$, $\max\{r_k, 1/r_h\}$ takes the minimum. That means the minimum elastic ratio can be achieved if and only if the ratio of each $e \in E$ takes the same value. □

Now we turn to the main theorem.

Theorem 2. *We can solve the elastic linkage problem from path to path in* $O(n^3)$ *time.*

Proof. We assume path $P = (v_1, v_2, \ldots, v_n)$, the length of each edge $\{v_i, v_{i+1}\}$ is l_i, path $G = (u_1, u_2, \ldots, u_{n'})$, the length of each edge $\{u_j, u_{j+1}\}$ is w_j, and $n \geq n' \geq 2$.

We define two functions as follows for $i > i' \geq j$:

$$dist(v_{i'}, v_i) = l_{i'} + l_{i'+1} + \cdots + l_{i-1}, \text{ and}$$
$$Ser(v_{i'}, v_i, w_j) = \max\left\{\frac{w_j}{dist(v_{i'}, v_i)}, \frac{dist(v_{i'}, v_i)}{w_j}\right\}.$$

That is, $dist(v_{i'}, v_i)$ is the length of the path $(v'_i, \ldots, v_i)$, and $Ser(v_{i'}, v_i, w_j)$ is the minimum elastic ratio of all edges in the subpath $P' = (v_{i'}, v_{i'+1}, \ldots, v_i)$ of P that covers the edge $\{u_j, u_{j+1}\}$. We first precompute these functions as tables which will be referred in our polynomial-time algorithm. The computation of the corresponding table $Ser[(v_{i'}, v_i), w_j]$ can be done as follows[1]: (1) for each $(v_{i'}, v_i)$ with $i' < i$, compute $dist(v_{i'}, v_i)$ and fill in the table $dist[v_{i'}, v_i]$, (2) for each $j = 0, 1, \ldots, n'$, compute $Ser(v_{i'}, v_i, w_j)$ and fill in the table $Ser[(v_{i'}, v_i), w_j]$. In (1), each $dist(v_{i'}, v_i)$ can be computed in a constant time by using $dist(v_{i'}, v_i) = dist(v_{i'}, v_{i-1}) + \ell(e_{i-1})$ when we compute the values of this table in the order of $(i - i') = 1, 2, 3, \ldots$. On the other hand, in (2), each $Ser(v_{i'}, v_i, w_j)$ can be computed in a constant time. Therefore, the precomputation can be done in $O(n^3)$ time in total.

To solve the elastic linkage problem efficiently, we define two more functions $ER(v_i, u_j)$ and $M(v_i, u_j)$ as follows. First, $ER(v_i, u_j)$ is the minimum elastic ratio of the mappings from the subpath $P' = (v_1, v_2, \ldots, v_i)$ of P to the subpath $G' = (u_1, u_2, \ldots, u_j)$ of G. Then we have the following:

$$ER(v_i, u_j) = \begin{cases} Ser(v_1, v_i, w_1) & \text{when } j = 2 \\ \min_{j-1 \leq k \leq i-1}\{\max\{ER(v_k, u_{j-1}), Ser(v_k, v_i, w_{j-1})\}\} & \text{when } j > 2 \end{cases}$$

Our goal is to obtain the mapping from P to G with elastic ratio $ER(v_n, u_{n'})$.

[1] In this paper, for a function $f()$ and a predicate $p()$, their corresponding tables (or arrays in program) are denoted by $f[]$ and $p[]$, respectively.

Next, $M(v_i, u_j)$ is a sequence of j vertices of path P that represents the mapping with minimum elastic ratio from the subpath P' to the subpath G'. The first and last vertices in $M(v_i, u_j)$ are v_1 and v_i. Then we have the following:

$$M(v_i, u_j) = \begin{cases} (v_1, v_i) & \text{when } j = 2 \\ (M(v_\tau, u_{j-1}), v_i) & \text{when } j > 2, \end{cases}$$

where τ is determined by the following equation;

$$ER(v_i, u_j) = \max\{ER(v_\tau, u_{j-1}), Ser(v_\tau, v_i, w_{j-1})\}.$$

Then our goal is to obtain $M(v_n, u_{n'})$. The $ER(v_n, u_{n'})$ and $M(v_n, u_{n'})$ can be obtained simultaneously by the dynamic programming technique. In the tables of $ER(v_n, u_{n'})$ and $M(v_n, u_{n'})$, $ER(v_n, u_{n'})$ and $M(v_n, u_{n'})$ are easy to get if the values in the $(n' - 2)$-nd row are available. The table $ER(v_n, u_{n'})$ is filled from $j = 2$, that is, for each $v_i = v_1, v_2, \ldots, v_n$, $ER(v_i, u_1) = Ser(v_1, v_i, w_1)$ has already been computed, and accordingly, the first row of table $M(v_n, u_{n'})$ is $M(v_i, u_1) = (v_1, v_i)$. After filling in the first row of the tables, it is easy to get the values in the second row, the third row, up to the $(n' - 2)$-nd row and finally get $ER(v_n, u_{n'})$ and $M(v_n, u_{n'})$. Each element of the table $ER(v_n, u_{n'})$ can be computed in $O(n)$ time, and each element of the table $M(v_n, u_{n'})$ can be computed in constant time. Therefore, the computation of $ER(v_n, u_{n'})$ and $M(v_n, u_{n'})$ can be done in $O(n^3)$ time, and the precomputation also can be done in $O(n^3)$ time. Thus, the algorithm runs in $O(n^3)$ time, which means the elastic linkage problem can be solved in polynomial time. □

5 Traverse Problem of a Tree by a Path

In this section, we focus on the traverse problem of G by P. In this variant, we allow P to cover an edge of G twice.

Before the traverse problem, we consider more general case that allow P to cover an edge of G twice or more. This general simulation problem is similar to the following ruler folding problem:

Ruler Folding: Given a polygonal chain with links of integer length $\ell_1, \ldots, \ell_{n-1}$ and an integer L, can the chain be folded flat so that its total folded length is L?

The details of this problem and related results can be found in [3]. In our context, we have the following theorem:

Theorem 3. *The general simulation problem of G by P is NP-complete even if G is an edge.*

Proof. We can reduce the ruler folding problem to our problem by just letting G be an edge of length L. □

We note that the ruler folding problem is weakly NP-complete, and we have a simple pseudo-polynomial-time algorithm that runs in $O(nL)$ time as follows;

Input: Set of integers $S = \{\ell_1, \ldots, \ell_{n-1}\}$ and an integer L
Output: Determine if there is $I \subseteq \{1, \ldots, n-1\}$ with $\sum_{i \in I} \ell_i == L$
begin
  Initialize array $a[0], \ldots, a[L]$ by 0;
  Set $a[0] = 1$;
  foreach $i = 1, \ldots, n-1$ **do**
    foreach $j = 0, \ldots, L$ **do**
      if $a[j] == 1$ *and* $j + \ell_i \leq L$ **then** $a[j + \ell_i] = 1$
    end
  end
  if $a[L] == 1$ **then** output "Yes";
  else output "No"
end

Now we turn to the traverse problem. Even if a connected graph G has no Eulerian path, when we allow to visit each edge in G twice, we can visit all vertices of G by a path in the depth first search manner. Therefore, we consider the following *traversal problem* as a kind of the robot arm simulation problem:

Input: A path $P = (V, E)$ that forms a path $(v_1, v_2, \ldots, v_n)$, and a graph $G = (V', E')$ with length function $\ell : E \cup E' \to \mathbb{R}$.
Output: A mapping ϕ from P to G such that each edge in G is mapped from exactly two subpaths of P, or "No" if it does not exist.

We first observe that it is linear time solvable when each edge has the unit length just by depth first search. Therefore, it is an interesting question that asks the computational complexity when ℓ maps to few distinct values, especially, ℓ maps to two distinct values.

We give three hardness results about the traversal problem even if the graph G is a simple tree T and the edge lengths are quite restricted.

Theorem 4. *The traversal problem of a tree T by a path P is strongly NP-complete in each of the following cases: (1) T is a star $K_{1,n-1}$, and P consists of edges of two different lengths. (2) T is a spider, and all edges in G and P are of length p and q, where (2a) p and q are any two positive integers that are relatively prime, or (2b) $p = 1$ and $q = 2$.*

Proof. Since it is clear that each of the problems is in NP, we show their hardness. We will give polynomial-time reductions from the 3-Partition to our problems.
(1) T is a star $K_{1,4m+1}$. Among $4m+1$ edges, the length of $m+1$ edges is B, and the other $3m$ edges have length a_i for each $i = 1, 2, \ldots, 3m$ (Fig. 3). The construction of P is as follows. Let P' be a path that consists of $2B$ edges of length 1, and P'' be a path that consists of 2 edges of length B. Then the path P is obtained by joining $m + 1$ subpaths P'' and m subpaths P' alternatingly, that is, P is constructed by joining $P'', P', P'', P', P'', \ldots, P', P''$ as shown in Fig. 3.

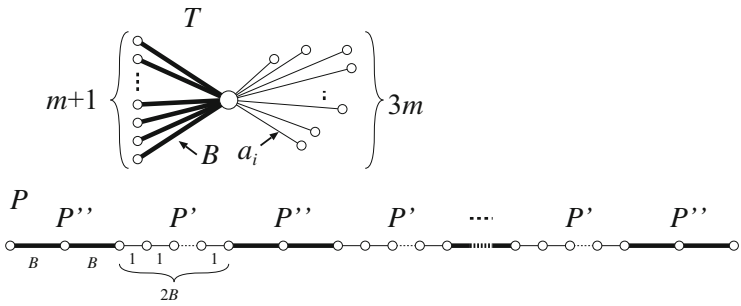


Fig. 3. Reduction to $K_{1,4m+1}$ and a path P.

The construction is done in polynomial time. Thus, we show that the 3-Partition problem has a solution if and only if the constructed cover problem has a solution. We first observe that P'' cannot cover any short edge of length a_i in T. Therefore, each P'' should cover each edge of length B in T twice. Hence all of the endpoints of P''s (and hence P') are on the central vertex of T. Therefore, if P can cover T properly, it is easy to see that each P' should cover three edges of length a_i, a_j, and a_k with $a_i + a_j + a_k = B$ exactly twice. This concludes the proof of (1).
(2a) This reduction is similar to (1). Let P' be a path that consists of $2B$ edges of length p, and P'' be a path that consists of 2 edges of length q. Then the path P is obtained by joining $m+1$ subpaths P'' and m subpaths P' alternatingly. On the other hand, the spider T is obtained by sharing the central vertex of $4m+1$ subpaths (Fig. 4). Among $4m+1$ subpaths, $m+1$ paths are just edges of length q. The other $3m$ subpaths are of a_i edges for each $1 \leq i \leq 3m$, and each edge has length p. Since p and q are relatively prime, P'' cannot cover each of the edges of length p. Therefore, their endpoints (and the endpoints of P') share the central vertex of T. Thus, each P' gives us the solution of the 3-Partition as in (1), which completes the proof of (2a).
(2b) The reduction itself is the same as (2a) except $p = 1$ and $q = 2$. In this case, we observe that no edge of length 1 can be covered by any edge of length 2 in P''. Therefore, each edge of P'' of length 2 should cover the edges of T of length 2. Thus, each P' gives us the solution of the 3-Partition as in (2a), which completes the proof of (2b). □

In Theorem 4, we show that the traversal problem of a tree by a path is NP-hard even if we strictly restrict ourselves. Now we turn to show a polynomial-time algorithm for the case that we furthermore restrict.

Theorem 5. *Let T be a star $K_{1,n'}$ and the number of distinct lengths of its edges is k. Let P be any path of length n. Without loss of generality, we suppose $2n' \leq n$. Then the traversal problem of T by P can be solved in $O(n^{k+1})$ time and $O(n^k)$ space. That is, it is polynomial time solvable when k is a constant.*

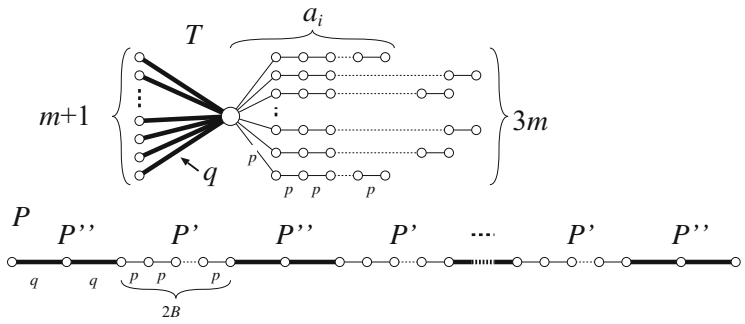


Fig. 4. Reduction to spider of two different lengths.

Proof. We suppose that each edge of T has a length in $L = \{\ell_1, \ell_2, \ldots, \ell_k\}$, and T contains L_i edges of length ℓ_i for each i. For a vertex v_i in P and length ℓ_j in L, we define a function $\mathrm{pre}(v_i, \ell_j)$ as follows;

$$\mathrm{pre}(v_i, \ell_j) = \begin{cases} v_k & \text{there is a vertex } v_k \text{ with } k < i \text{ on } P \text{ s. t. } \ell(e_k) + \cdots + \ell(e_i) = \ell_j, \\ \phi & \text{otherwise.} \end{cases}$$

We first precompute this function as a table which will be referred to in our polynomial-time algorithm. To distinguish the function $\mathrm{pre}(v_i, \ell_j)$, we refer to this table as $\mathrm{pre}[v_i, \ell_j]$ which uses $O(nk)$ space. The computation of pre[] can be done as follows; (0) initialize pre[] by ϕ in $O(nk)$ time, (1) sort L in $O(k \log k)$ time, and (2) for each $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, k-1$, the vertex v_i fills the table $\mathrm{pre}[v_{i'}, \ell_j] = v_i$. In (2), the vertex v_i can fill $\mathrm{pre}[v_{i'}, \ell_j] = v_i$ in $O(n+k)$ time. Therefore, the precomputing takes $O(n(n+k) + k \log k)$ time in $O(nk)$ space.

Now we turn to the computation for the traversal problem. To do that, we define a predicate $F(d_1, d_2, \ldots, d_k, v_i)$ which is defined as follows: When there is a cover of a subtree T' of T that consists of d_1 edges of length ℓ_1, d_2 edges of length ℓ_2, ..., and d_k edges of length ℓ_k by the subpath $P' = (v_1, v_2, \ldots, v_i)$ when v_1 and v_i are put on the center of T, $F(d_1, d_2, \ldots, d_k, v_i)$ is *true*, and *false* otherwise. (For notational convenience, we define that $F(d_1, d_2, \ldots, d_k, \phi)$ is always false.) Thus, our goal is to determine if $F(L_1, L_2, \ldots, L_k, v_n)$ is true or false. The predicate $F(d_1, d_2, \ldots, d_k, v_i)$ is determined by the following recursion;

$$F(d_1, d_2, \ldots, d_k, v_i) = \bigvee_{1 \le j \le k} ((\mathrm{pre}(v_i, \ell_j) \neq \phi) \wedge F(d_1, \ldots, d_j - 2, \ldots, d_k, \mathrm{pre}(\mathrm{pre}(v_i, \ell_j), \ell_j))).$$

That is, for the vertex v_i, we have to have two vertices $v_{i'} = \mathrm{pre}(v_i, \ell_j)$ and $v_{i''} = \mathrm{pre}(\mathrm{pre}(v_i, \ell_j), \ell_j)$ such that $\ell(e_{i'}) + \ell(e_{i'+1}) + \cdots + \ell(e_i) = \ell_j$ and $\ell(e_{i''}) + \ell(e_{i''+1}) + \cdots + \ell(e_{i'}) = \ell_j$ for some j with $1 \le j \le k$. The correctness of this recursion is trivial.

The predicate $F(L_1, L_2, \ldots, L_k, v_n)$ is computed by a dynamic programming technique. That is, the table $F[d_1, d_2, \ldots, d_k, v_i]$, corresponding to the predicate $F(L_1, L_2, \ldots, L_k, v_n)$, is filled from $d_1 = 0, d_2 = 0, \ldots, d_k = 0$ for the center vertex c, which is true. Then, we increment in the bottom up manner; that is, we increment as $(d_1, d_2, \ldots, d_k) = (0, 0, \ldots, 0, 1)$, $(0, 0, \ldots, 1, 0)$, $\ldots$, $(0, 1, \ldots, 0, 0)$, $(1, 0, \ldots, 0, 0)$, $(0, 0, \ldots, 0, 2)$, $(0, 0, \ldots, 1, 1)$, $\ldots$, $(0, 1, \ldots, 0, 1)$, $(1, 0, \ldots, 0, 1)$, and so on. The number of combinations of $(d_1, d_2, \ldots, d_k)$ is $L_1 \cdot L_2 \cdot \cdots \cdot L_k \leq n'^k = O(n^k)$, and the computation of $F[d_1, d_2, \ldots, d_k, v_i]$ for the $(d_1, d_2, \ldots, d_k)$ can be done in linear time. Therefore, the algorithm runs in $O(n^{k+1})$ time and $O(n^k)$ space. □

References

1. Demaine, E.D., O'Rourke, J.: Geometric Folding Algorithms. Cambridge University Press, Cambridge (2007)
2. Kupferman, O., Vardi, G.: Eulerian paths with regular constraints. In: 41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016). Leibniz International Proceedings in Informatics (LIPIcs), vol. 58, 62:1–62:15 (2016)
3. Michael, R.G., David, S.J.: Computers and Intractability: A Guide to the Theory of NP-Completeness, pp. 90–91. W. H. Freeman Co., San Francisco (1979)

Evaluation of Tie-Breaking and Parameter Ordering for the IPO Family of Algorithms Used in Covering Array Generation

Kristoffer Kleine[1], Ilias Kotsireas[2], and Dimitris E. Simos[1(✉)]

[1] SBA Research, 1040 Vienna, Austria
{kkleine,dsimos}@sba-research.org
[2] Wilfrid Laurier University, Waterloo, ON, Canada
ikotsire@wlu.ca

Abstract. The IPO (In-Parameter-Order) family of algorithms is a popular set of greedy methods for the construction of covering arrays. Aspects such as tie-breaking behavior or parameter ordering can have major impact on the quality of the resulting arrays but have so far not been studied in a systematic manner. In this paper, we survey and present a general framework for the IPO family of algorithms (i.e. IPOG, IPOG-F and IPOG-F2) and present ways to instantiate these abstract components. Then, we evaluate the performance of these variations on a large set of instances, in an extensive experimental setting in terms of covering array sizes.

Keywords: Covering arrays · IPO family · Tie-breaks
Parameter ordering · Experiments

1 Introduction

A covering array (CA) is a mathematical object defined by four positive integers and denoted as $CA(N; t, k, v)$. It is a $N \times k$ matrix where N is the number of rows, k the number of columns (often referred to as parameters), t the size of interactions that are covered and v is the size of the alphabet. A covering array is defined by its *t-covering property*: for any t-selection of columns, all v^t t-tuples between the selected columns occur at least once in the array. t is called the strength of the CA. A mixed-level covering array (MCA) is a generalization of a CA where each column i has its own alphabet size v_i. An MCA is denoted as $MCA(N; t, k, (v_1, \ldots, v_k))$. The tuple $(t, k, (v_1, \ldots, v_k))$ $(t \leq k)$ is referred to as the configuration of an MCA.

The general problem of constructing optimal covering arrays (i.e., in terms of minimal size N) is believed to be a hard combinatorial optimization problem and it has significant applications in software and hardware testing [8,9]. Moreover, it is tightly coupled with NP-hard problems [2]. As a result, there has been a lot

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 189–200, 2018.
https://doi.org/10.1007/978-3-319-94667-2_16

of effort on developing and improving algorithmic approaches for covering array generation (c.f. Sect. 2).

The In-Parameter-Order (IPO) family of algorithms is a set of greedy algorithms for constructing covering arrays and their representatives have been shown to produce acceptable sized covering arrays [3]. There have been multiple efforts in the area of improving the IPO variants, in terms of reducing the size of generated covering arrays, but to the best of our knowledge, no systematic evaluation of these proposals exist. In this work, we aim to provide an overview of these optimization efforts and evaluate their effectiveness.

This paper is structured as follows: In Sect. 2 we discuss preliminaries and related work. Section 3 presents the IPO family of algorithms and existing and novel methods to parameterize it. Section 4 proposes an evaluation of the algorithms and discusses the results and finally, Sect. 5 concludes the paper.

2 Background

The In-Parameter-Order (IPO) strategy was first proposed in [15] as a greedy algorithm for covering array construction. It constructs a covering array one column at a time and each extension is divided into a horizontal and an optional vertical extension. We discuss the algorithm in greater detail in Sect. 3.

While the original algorithm was limited to arrays of strength 2 (pair-wise), subsequent works have generalized the algorithm to allow the generation of higher strength arrays [13] (IPOG) as well as integrated constraint handling in [17,18]. In [5], the authors propose variants of the IPO strategy, namely IPOG-F, IPOG-F2, by extending the search space in the horizontal extension. In [14], the IPOG-D variant is presented which includes a recursive construction method aimed at reducing the number of combinations to be enumerated.

Many works have been dedicated to improving parts of the IPO algorithms in order to minimize covering arrays sizes. In [4] a graph-coloring scheme integrated into the vertical extension is proposed to reduce the resulting array sizes. [16] modify IPOG with additional optimizations aimed at reducing *don't-care* values in order to minimize the number of rows. [6,7] discuss and evaluate the impact of tie-breaking on the generated arrays and propose a new tie-breaker which reduces the generated array sizes.

Related to the aim of this paper is the works of [1] for "one-test-at-a-time" greedy construction algorithms. There, the authors propose a general framework of such algorithms and provide concrete ways to instantiate it. In their analysis they evaluate the most important parameters which contribute to smaller covering array sizes.

3 The IPO Family of Algorithms

The IPO family of algorithms is a set of algorithms for constructing covering arrays and has seen use in many applications in practice [12]. The most important representatives are IPOG [13], IPOG-F [5] and IPOG-F2 [5]. They are greedy

algorithms able to generate (mixed-level) covering arrays for any configuration. They all follow the same basic schema (c.f. Algorithm 1) and start out by building an initial covering array of strength t by constructing the cross-product of the first t columns.

Algorithm 1. IPO Algorithm

```
procedure IPO(configuration)
    configuration ← Sort-Criterion(configuration)
    Array ← cross-product of first t columns
    for i ← t, ..., k do
        HorizontalExtension(i)
        if there are uncovered tuples then
            VerticalExtension(i)
        end if
    end for
end procedure
```

Then, the array is extended one column at a time in a two-dimensional fashion. First, a new column is added to the array by a horizontal extension. This step will assign values to the new column in a greedy manner with the objective to maximize coverage gain. Algorithm 2 shows the procedure for `IPOG`. Here, rows are considered from top to bottom and in each one the value with the maximum coverage gain (in terms of previously uncovered t-tuples) is selected. If multiple values provide maximum gain then the tie must be broken with a predefined strategy. This will be discussed in further detail in Sect. 3.1.

`IPOG-F` and `IPOG-F2` extend the search space in the horizontal extension by also optimizing the order in which rows are extended. For this purpose, unassigned rows are additionally scanned in each iteration to find the best position and value to assign. While `IPOG-F` considers the actual coverage to decide on the best value, `IPOG-F2` tries to estimate the amount of covered tuples. This is an optimization which allows for a faster implementation compared to `IPOG-F`, but it sacrifices accuracy. The estimation is achieved by keeping an estimated value of covered tuples in an array for each row and value. When a value a is chosen in row r, the estimator is incremented by the number of shared tuples between row s and all other unassigned rows s.

The horizontal extension either finishes when all tuples have been covered or each entry in the new column has been assigned a value. In the latter case, there are still tuples left which have not yet been covered and they are added in the vertical extension step. This step grows the array by adding more rows which include missing tuples.

In the vertical extension all remaining uncovered tuples are added to the array to ensure that the first i columns form a covering array. Tuples can either be added by appending a new row to the array that contains the tuple or by finding an already existing row which can fit the tuple. The latter case is possible

Algorithm 2. Horizontal Extension (IPOG)

```
procedure HorizontalExtension(i)
    for row ← 0, ... ,Array.rows do
        v ← select value with highest coverage gain
        if multiple candidate values then
            break tie
        end if
        Array[row][i] ← v
        if all tuples are covered then return
        end if
    end for
end procedure
```

since don't-care values can occur in the array and may be overwritten by a tuple without destroying the t-covering property.

3.1 Tie-Breakers

During the horizontal extension, tie-breaking may be necessary in the case that two or more values for a row in the new column provide the same, maximum coverage gain, i.e. cover the most new t-tuples. We will refer to these values as candidates. In the following we will give an overview to possible tie-breaking strategies.

Random Tie-Breaker. The simplest approach is to choose one value out of all candidates at random. This can be implemented efficiently but it will introduce non-determinism to algorithm and the generated covering arrays will possibly differ on subsequent runs of the algorithm. This tie-breaker is oblivious to the previous history of the extension.

Deterministically-Seeded Random Tie-Breaker. This is a variant of the `Random` tie-breaker. Here, ties are still broken randomly with the help of a pseudorandom generator, but the generator is seeded with a constant at the beginning which results in a deterministic behaviour of the algorithm.

Lexicographic Tie-Breaker. This tie-breaker will always prefer the (lexicographically) smallest candidate if multiple are available. This can of course introduce a bias towards smaller values in the new column.

Cyclic Tie-Breaker. This tie-breaker builds upon the `Lexicographic` tie-breaker, but maintains the last chosen value and starts the search from this one instead of the first. The aim is to remove bias towards smaller values, however the last chosen value is more likely to be picked again in the next iteration.

Cyclic-Next Tie-Breaker. This tie-breaker works exactly as the `Cyclic` tie-breaker, but will start from the next value following the last chosen value. This tie-breaker was first proposed in [6,7].

Value-Balanced Tie-Breaker. This tie-breaker keeps track of how many times a value has been used so far in the extended column. In an optimal situation, each value for the new column occurs exactly the same amount of times and the aim of this tie-breaker is to mimic this behaviour by balancing the occurrences of these values. Values are preferred when they so far have occured less frequently than other candidate values.

α-balanced Tie-Breaker. This tie-breaker builds upon the value-balanced tie-breaker by not only considering the balance of values in the new column, but the balance of lower-strength tuples involving the new parameter. This is based on the notion of α-balance which was introduced by [10] and functions as a tie-breaker in the following way: first, the number of would-be-covered $t-1$ tuples are compared for each candidate. If there still is a tie, the next lower strength is tried and so on. If at $t=1$ there still exists a tie, then the smallest value will be preferred.

3.2 Tuple Enumeration Order

In the vertical extension, uncovered tuples are added one-by-one to the array. So far it has not been studied, if different enumeration orders of these tuples have any impact on the resulting arrays. We propose besides the common lexicographically-ascending (tuples of small lexicographical order first) order, the reverse, i.e. from lexicographically largest to smallest. Furthermore, switching between the orderings every other vertical extension could prove beneficial to achieve smaller arrays.

3.3 Parameter Ordering

One simple option to influence the covering array generation is the order in which columns are extended. Since the covering property is not affected by column permutations one can permute the configuration before starting the generation and apply the reverse permutation afterwards. Note that this is only useful for mixed-level covering arrays. Informal consensus is that the `IPO` strategy generates smaller arrays when columns are sorted by decreasing alphabet size, but, to the best of our knowledge, this has so far not been subject to an experimental evaluation.

While the number of column permutations in general is too large in practice, we propose to investigate the following:

Ascending Sort columns with increasing alphabet size from smallest to largest
Descending Sort columns with decreasing alphabet size from largest to smallest

Alternating Intersperse large and small columns and switch between large and small columns from one extension to the next. We propose two variants. The first starts with the smallest, followed by the largest and thirdly the second-smallest, etc. The second starts with the largest, followed by the smallest and so on.

4 Evaluation

4.1 Setup

To evaluate the different algorithm configurations we chose a set of (M)CA instances based upon the benchmarks used in [1] to study the behaviour of greedy, one-test-at-a-time MCA generation algorithms. The instances are summarized in Fig. 1a.

Instances
10^4
3^{40}
3^4
6^4
$3^4, 4^5$
$6^6, 5^5, 3^4$
$7^8, 2^{20}$
$5^1, 3^8, 2^2$
$5^{10}, 2^{10}$
$8^2, 7^2, 6^2, 5^2$
$10^1, 9^1, 8^1, 7^1, 6^1, 5^1, 4^1, 3^1, 2^1, 1^1$

(a) Set of benchmark (M)CA instances

Tie Breakers	Tuple Orders	Parameter Orders
Alpha-Balanced	Alternating	Alternating-large
Cyclic	Ascending	Alternating-small
Cyclic-next	Descending	Ascending
Deterministic		Descending
Lexicographic		
Random		
Value-Balanced		

(b) Configuration options for `IPO`

Fig. 1. Benchmark setup

We implemented all IPO variants in our own implementation described further in [11]. The particular algorithm as well as the tie-breaker, tuple order and parameter ordering are selectable via a configuration option at runtime. This results in 63 distinct algorithm configurations for CA generation and 252 distinct configurations for MCA generation. The configuration options are summarized in Fig. 1b.

Each algorithm configuration was used to generate (M)CAs for the selected benchmark instances for strengths between 2 and 4. The experiments were conducted on the Graham cluster of the Shared Hierarchical Academic Research Computing Network (SHARCNET). Configurations involving the `Random` tie-breaker were repeated 10 times. Due to space limitations, we only discuss selected and aggregated results, but we provide the full data set as well as visualizations

on a dedicated website[1] for the interested reader. There we also provide further measurements into the test generation time differences observed between the tested configurations.

4.2 Results

In order to meaningfully compare different configurations options across instances we first normalized the computed covering array sizes to a relative measure representing the deviation of the mean. We computed the mean for each instance and based on the result computed the relative improvement or degradation for each individual run. This value shows how much better or worse one configuration performs in comparison to the other ones. The results are summarized in Table 1 and are visualized in Figs. 2a and b.

Table 1. Relative improvement for different configurations compared to the mean

	IPOG	IPOG-F	IPOG-F2
Tie Breaker			
Alpha-balanced	1.0128 ±0.0759	0.9632 ±0.0619	1.0572 ±0.0662
Cyclic	1.0175 ±0.1394	0.9461 ±0.0678	1.0379 ±0.0867
Cyclic-next	0.9721 ±0.0858	0.9403 ±0.0799	1.0296 ±0.1012
Deterministic	0.9920 ±0.0429	0.9560 ±0.0539	1.0500 ±0.0664
Lexicographic	1.0140 ±0.0764	0.9651 ±0.0687	1.0580 ±0.0673
Random	0.9933 ±0.0465	0.9548 ±0.0545	1.0497 ±0.0673
Value-balanced	0.9951 ±0.0630	0.9590 ±0.0563	1.0549 ±0.0645
Tuple Order			
Alternating	0.9950 ±0.0656	0.9533 ±0.0580	1.0454 ±0.0681
Ascending	0.9987 ±0.0663	0.9582 ±0.0626	1.0584 ±0.0794
Descending	0.9944 ±0.0630	0.9530 ±0.0561	1.0432 ±0.0644
Parameter Order			
Alternating-large	0.9962 ±0.0309	0.9529 ±0.0268	1.0503 ±0.0510
Alternating-small	1.0082 ±0.0374	0.9667 ±0.0336	1.0763 ±0.0525
Ascending	1.0285 ±0.0443	1.0006 ±0.0440	1.0870 ±0.0519
Descending	0.9463 ±0.0791	0.8910 ±0.0442	0.9952 ±0.0793

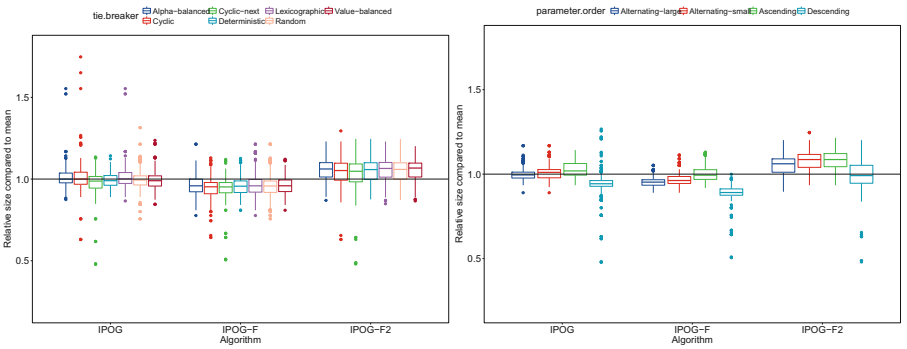
In general, `IPOG-F` produces the smallest arrays, followed by `IPOG` and `IPOG-F2`. Comparing the results for the different tie-breakers, no one choice seems to impact array sizes significantly, however, the `Cyclic-next` tie-breaker overall yields the best results. It, together with the `Cyclic` tie-breaker is able to generate some arrays with up to 50% less rows. However, the `Cyclic` tie-breaker exhibits extreme results in the other direction and in corner-cases with array

[1] https://matris.sba-research.org/data/iwoca2018.

sizes exceeding 50% larger than the mean are produced. This is also the case for the `Alpha-balanced` and `Lexicographic` tie-breaker.

Judging from the results in Table 1, the order in which tuples are enumerated does not seem to affect the resulting covering array size in any significant way.

The largest impact can be attributed to the sorting order of columns. Sorting in descending order of alphabet size leads to significantly smaller covering arrays, especially in the case of `IPOG-F`. Alternating between large and small columns has some impact and is better than sorting columns in ascending order.



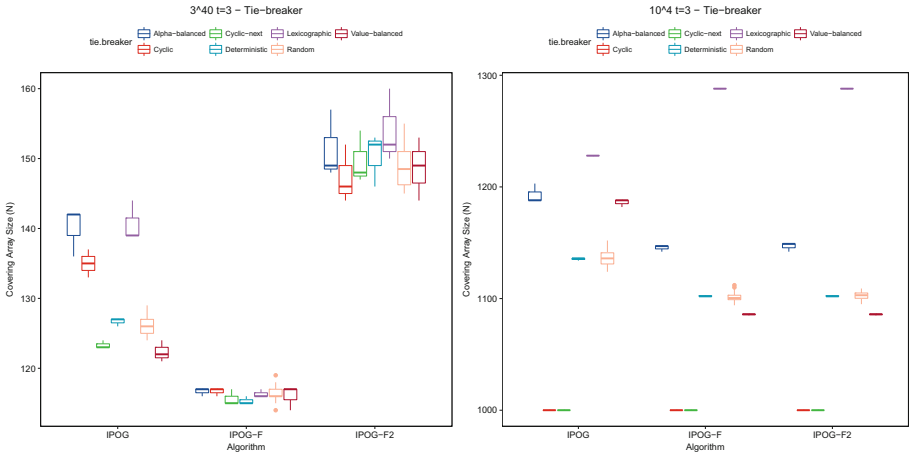
(a) Tie-breakers (b) Parameter orders

Fig. 2. Relative improvement compared to the mean

Selected benchmark results. Aside from the general performance, for specific instances the various configuration options can have differing impact. In the following, we discuss some results for selected instances. In order to meaningfully analyze the results we have grouped the results by both the algorithm (i.e., `IPOG`, `IPOG-F` or `IPOG-F2`) and one of either tie-breaker, tuple-order or parameter-order. Inside each group we have computed the mean and the standard deviation. The results show absolute values instead of relative difference.

3^{40} ($t = 3$) The results of this experiment are summarized in Table 2 and the generated covering array sizes per tie-breaker are visualized in Fig. 3a. `IPOG-F` produces the smallest arrays and shows very low variance when comparing different tie-breakers. In contrast, the results for `IPOG` are much more dependent on the tie-breaker. Here, the best results are obtained with the `Value-balanced` tie-breaker which produces arrays 16% smaller than when using the `Alpha-balanced` tie-breaker. `IPOG-F2` shows no significantly differing behavior with different tie-breakers. Furthermore, the order in which tuples are enumerated have no major impact.

10^4 ($t = 3$) The results for this experiment can be found in Table 2 and a comparison of the tie breakers can be found in Fig. 3b. Here, the configurations which use either the `Cyclic` or `Resuming` tie-breakers manage to generate an



(a) 3^{40} $(t = 3)$ (b) 10^4 $(t = 3)$

Fig. 3. Results for different tie-breakers

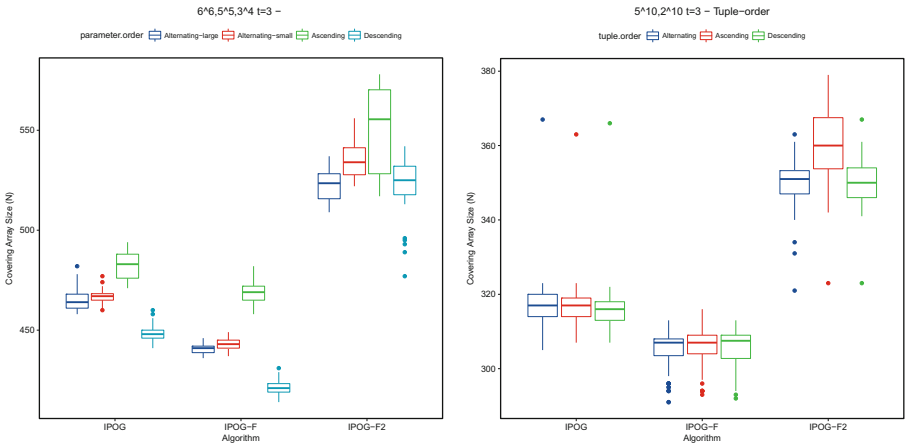
Table 2. Results for CA experiments

	3^{40} $t = 3$			10^4 $t = 3$		
	IPOG	IPOG-F	IPOG-F2	IPOG	IPOG-F	IPOG-F2
Tie Breaker						
Alpha-balanced	140.0 ±3.5	116.7 ±0.6	151.3 ±4.9	1193.0 ±8.7	1145.3 ±2.9	1146.7 ±4.0
Cyclic	135.0 ±2.0	116.7 ±0.6	147.3 ±4.2	1000.0 ±0.0	1000.0 ±0.0	1000.0 ±0.0
Cyclic-next	123.3 ±0.6	115.7 ±1.2	149.7 ±3.8	1000.0 ±0.0	1000.0 ±0.0	1000.0 ±0.0
Deterministic	126.7 ±0.6	115.3 ±0.6	150.3 ±3.8	1135.3 ±1.2	1102.3 ±0.6	1102.3 ±0.6
Lexicographic	140.7 ±2.9	116.3 ±0.6	154.0 ±5.3	1228.0 ±0.0	1288.0 ±0.0	1288.0 ±0.0
Random	125.9 ±1.3	116.4 ±1.0	148.9 ±2.7	1136.2 ±7.0	1101.4 ±4.3	1102.2 ±3.7
Value-balanced	122.3 ±1.5	116.0 ±1.7	148.7 ±4.5	1186.0 ±3.5	1085.7 ±0.6	1085.7 ±0.6
Tuple Order						
Alternating	127.6 ±5.7	116.2 ±0.9	147.4 ±2.3	1130.0 ±58.2	1102.4 ±62.1	1103.7 ±62.1
Ascending	128.7 ±6.4	116.5 ±1.0	152.6 ±3.0	1131.9 ±59.2	1102.1 ±61.9	1102.1 ±61.9
Descending	127.6 ±4.8	116.1 ±1.0	148.3 ±2.5	1132.6 ±58.2	1102.1 ±62.0	1102.6 ±62.2

orthogonal array (since the size is equal to v^t) for the three algorithms. Interestingly, the `Lexicographic` tie-breaker, although similar to the other two, performs the worst in all cases with almost 30% larger array sizes. As before, in this case the tuple order has no real impact.

4.3 $6^6 5^5 3^4$ $(t = 3)$

For this instance (see Table 3), there is no large variance when comparing different tie-breakers. `IPOG-F` produces the smallest arrays, while `IPOG-F2` produces the largest. Here, the parameter order has a measurable impact and ordering the parameters by descending size can improve array sizes by up to 5% in this case. These results are visualized in Fig. 4a.



(a) $6^6 5^5 3^4$ $(t = 3)$ (b) $5^{10} 2^{10}$ $(t = 3)$

Fig. 4. Results for different parameter orders (left) and tuple orders (right)

Table 3. Results for MCA experiments

	$6^6 5^5 3^4 t = 3$			$5^{10} 2^{10} t = 3$		
	IPOG	IPOG-F	IPOG-F2	IPOG	IPOG-F	IPOG-F2
Tie Breaker						
Alpha-balanced	470.5 ±13.6	441.9 ±17.1	533.8 ±18.2	316.0 ±5.2	305.5 ±5.4	353.5 ±7.8
Cyclic	465.8 ±14.1	443.9 ±16.3	532.3 ±16.4	331.1 ±20.7	308.2 ±1.6	344.2 ±14.0
Cyclic-next	464.7 ±12.3	444.0 ±18.1	529.0 ±23.8	316.3 ±4.4	307.6 ±3.9	352.0 ±11.8
Deterministic	465.2 ±13.9	442.8 ±20.0	532.1 ±15.8	316.2 ±3.5	304.1 ±6.0	355.0 ±7.7
Lexicographic	471.6 ±9.8	442.3 ±17.2	525.5 ±26.1	316.9 ±4.0	306.2 ±7.0	355.0 ±8.3
Random	465.2 ±13.2	443.5 ±17.3	533.8 ±16.9	315.4 ±4.1	304.6 ±5.5	353.8 ±8.5
Value-balanced	462.9 ±14.2	443.7 ±19.3	530.8 ±16.1	314.6 ±4.3	304.8 ±5.7	352.5 ±6.6
Tuple Order						
Alternating	466.2 ±14.0	442.5 ±17.9	527.6 ±18.7	316.6 ±7.9	304.7 ±5.6	349.8 ±6.6
Ascending	466.3 ±14.8	444.6 ±19.1	542.5 ±20.2	317.0 ±7.0	305.4 ±5.5	359.9 ±10.2
Descending	464.9 ±10.3	443.0 ±14.9	527.7 ±8.0	316.0 ±7.4	305.3 ±5.3	349.7 ±6.1
Parameter Order						
Alternating-large	465.1 ±5.1	440.4 ±2.6	522.5 ±8.0	317.8 ±2.4	307.2 ±2.4	348.3 ±3.7
Alternating-small	467.0 ±3.4	442.9 ±3.0	534.7 ±7.9	317.7 ±2.2	308.1 ±2.2	353.2 ±5.1
Ascending	482.8 ±6.3	468.8 ±5.4	550.8 ±20.7	317.6 ±2.6	307.5 ±2.5	360.4 ±9.3
Descending	448.4 ±4.1	421.4 ±3.6	522.4 ±13.9	313.2 ±13.8	297.7 ±5.4	350.7 ±11.4

$5^{10}2^{10}(t = 3)$ The results for this instance are described in Table 3 and a comparison of different tuple orders is visualized in Fig. 4b. The tuple order seems to only make a difference for `IPOG-F2`, where both the `Alternating` and `Descending` order outperform the `Ascending` order. This is also the case in instance $6^6 5^5 3^4$ $(t = 3)$.

5 Conclusion

In this paper we have studied the impact of tie-breaking, parameter ordering and tuple enumeration order in the `IPO` family of algorithms. We have compared their effectiveness in terms of their ability to reduce covering array sizes in a large evaluation. In summary, `IPOG-F` overall manages to produce the smallest arrays compared to `IPOG` and `IPOG-F2`. Furthermore, the choice of tie-breaker seems to not matter a great deal when averaging over all instances, but the right choice can have large impact on selected instances. In the case of MCA generation, we measured the largest reduction in array size when ordering columns by decreasing alphabet size, with up to 12% reduction in size compared to the mean.

Acknowledgments. The research presented in this paper has been funded in part by the Austrian Research Promotion Agency (FFG) under grants 851205 (Security Protocol Interaction Testing in Practice - SPLIT) and 865248 (SECuring Web technologies with combinatorial Interaction Testing - SecWIT).

Part of this research has also been carried out in the context of the Austrian COMET K1 program which is publicly funded by the Austrian Research Promotion Agency (FFG) and the Vienna Business Agency (WAW).

This work was made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET) and Compute/Calcul Canada.

References

1. Bryce, R.C., Colbourn, C.J., Cohen, M.B.: A framework of greedy methods for constructing interaction test suites. In: Proceedings of the 27th International Conference on Software Engineering, ICSE 2005, pp. 146–155. ACM (2005)
2. Cheng, C.T.: The test suite generation problem: optimal instances and their implications. Discrete Appl. Math. **155**(15), 1943–1957 (2007)
3. Cohen, M.B., Gibbons, P.B., Mugridge, W.B., Colbourn, C.J.: Constructing test suites for interaction testing. In: Proceedings of the 25th International Conference on Software Engineering, ICSE 2003, pp. 38–48. IEEE Computer Society (2003)
4. Duan, F., Lei, Y., Yu, L., Kacker, R.N., Kuhn, D.R.: Improving IPOG's vertical growth based on a graph coloring scheme. In: 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 1–8 (2015)
5. Forbes, M., Lawrence, J., Lei, Y., Kacker, R.N., Kuhn, D.R.: Refining the in-parameter-order strategy for constructing covering arrays. J. Res. Nat. Inst. Stan. Technol. **113**(5), 287 (2008)
6. Gao, S.W., Lv, J.H., Du, B.L., Colbourn, C.J., Ma, S.L.: Balancing frequencies and fault detection in the in-parameter-order algorithm. J. Comput. Sci. Technol. **30**(5), 957–968 (2015)
7. Gao, S., Lv, J., Du, B., Jiang, Y., Ma, S.: General optimization strategies for refining the in-parameter-order algorithm. In: 2014 14th International Conference on Quality Software (QSIC), pp. 21–26. IEEE (2014)
8. Hartman, A.: Software and hardware testing using combinatorial covering suites. In: Golumbic, M., Hartman, I.A. (eds.) Graph Theory, Combinatorics and Algorithms, Operations Research/Computer Science Interfaces Series, vol. 34, pp. 237–266. Springer, Heidelberg (2005)

9. Hartman, A., Raskin, L.: Problems and algorithms for covering arrays. Discrete Math. **284**(1–3), 149–156 (2004)
10. Kampel, L., Simos, D.E.: Set-based algorithms for combinatorial test set generation. In: Wotawa, F., Nica, M., Kushik, N. (eds.) ICTSS 2016. LNCS, vol. 9976, pp. 231–240. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47443-4_16
11. Kleine, K., Simos, D.E.: An efficient design and implementation of the in-parameter-order algorithm. Math. Comput. Sci. **12**(1), 51–67 (2018)
12. Kuhn, D., Kacker, R., Lei, Y.: Introduction to Combinatorial Testing. Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series. Taylor & Francis, London (2013)
13. Lei, Y., Kacker, R., Kuhn, D.R., Okun, V., Lawrence, J.: IPOG: a general strategy for T-way software testing. In: 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, ECBS 2007, pp. 549–556. IEEE (2007)
14. Lei, Y., Kacker, R., Kuhn, D.R., Okun, V., Lawrence, J.: IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing. Softw. Test. Verification Reliab. **18**(3), 125–148 (2008)
15. Lei, Y., Tai, K.C.: In-parameter-order: a test generation strategy for pairwise testing. In: Proceedings of Third IEEE International High-Assurance Systems Engineering Symposium, pp. 254–261. IEEE (1998)
16. Younis, M.I., Zamli, K.Z.: MIPOG-an efficient t-way minimization strategy for combinatorial testing. Int. J. Comput. Theory Eng. **3**(3), 388 (2011)
17. Yu, L., Duan, F., Lei, Y., Kacker, R.N., Kuhn, D.R.: Constraint handling in combinatorial test generation using forbidden tuples. In: 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 1–9 (2015)
18. Yu, L., Lei, Y., Nourozborazjany, M., Kacker, R.N., Kuhn, D.R.: An efficient algorithm for constraint handling in combinatorial test generation. In: 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, pp. 242–251 (2013)

Efficient Enumeration of Subgraphs and Induced Subgraphs with Bounded Girth

Kazuhiro Kurita[1(✉)], Kunihiro Wasa[2], Alessio Conte[2], Takeaki Uno[2], and Hiroki Arimura[1]

[1] IST, Hokkaido University, Sapporo, Japan
{k-kurita,arim}@ist.hokudai.ac.jp
[2] National Institute of Informatics, Tokyo, Japan
{wasa,conte,uno}@nii.ac.jp

Abstract. The girth of a graph is the length of its shortest cycle. Due to its relevance in graph theory, network analysis and practical fields such as distributed computing, girth-related problems have been object of attention in both past and recent literature. In this paper, we consider the problem of listing connected subgraphs with bounded girth. As a large girth is index of sparsity, this allows to extract sparse structures from the input graph. We propose two algorithms, for enumerating respectively vertex induced subgraphs and edge induced subgraphs with bounded girth, both running in $O(n)$ amortized time per solution and using $O(n^3)$ space. Furthermore, the algorithms can be easily adapted to relax the connectivity requirement and to deal with weighted graphs. As a byproduct, the second algorithm can be used to answer the well known question of finding the densest n-vertex graph(s) of girth k.

1 Introduction

We consider the problem of finding all subgraphs and induced subgraphs with girth at least k of a graph. The girth is a measure of sparsity, as graphs with large girth are inherently sparse. This corresponds to finding *sparse* substructures of the given graph, a problem that was considered under several forms [5,9] and has applications in network analysis. In particular, this problem generalizes two well studied problems, i.e., listing all subtrees and induced subtrees [7,13–15]. Indeed, any graph with girth larger than n may not contain a cycle, i.e., it is a tree, or a forest.

A *subgraph enumeration problem*, given a graph G and some constraint $\mathcal{R}$, consists in outputting all the subgraphs satisfying $\mathcal{R}$ without duplicates. The efficiency of enumeration algorithms is often measured with respect to both the size of the input and that of the output, i.e., the number of solutions: an enumeration algorithm is called an *amortized polynomial time algorithm* if it runs in $O(M \cdot poly(N))$ time, where N is the input size and M is the number of solutions. Furthermore, the algorithm is said to have polynomial *delay* if the maximum time elapsed between two consecutive outputs is polynomial.

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 201–213, 2018.
https://doi.org/10.1007/978-3-319-94667-2_17

In this paper, we present two amortized polynomial time algorithms for enumerating subgraphs of girth at least k. The first, `EBG-IS`, enumerates *induced* subgraphs, while the second, `EBG-S`, enumerates *edge* subgraphs (also simply called subgraphs). Both `EBG-IS` and `EBG-S` run in $O(n\,|\mathcal{S}|)$ time using $O(n^3)$ space, where n is the number of nodes in G and $\mathcal{S}$ is the set of all solutions. The proposed algorithms will consider the enumeration of *connected* subgraphs in simple graphs. However, both algorithms can easily be applied to the enumeration of non-connected subgraphs, and to weighted graphs by trivial changes, with the same time and space complexity. In these problems, the upper bound of the number of solutions are $O(2^n)$ and $O(2^m)$, respectively, where m is the number of edges. Hence, the brute force algorithms are optimal if we evaluate the efficiency of algorithms only the input size. When we describe a more efficient algorithm, reducing amortized complexity is important [10]. Indeed, our implementation of `EBG-S`[1] is almost 560 times faster than the brute force algorithm when the input graph is a complete graph K_8 and girth is four.

While the problem of efficiently enumerating subgraphs with bounded girth has been considered for *directed* graphs [6], to the best of our knowledge, there is no known efficient algorithm for the *undirected* version of the problem.[2]

An early result on girth computation is the algorithm by Itai and Rodeh [8], that finds the girth of a graph in $O(nm)$ time. In more recent work, the problem was also solved in linear time for planar graphs [4]. However, the problem we consider involves computing the girth of many subgraphs, so relying on these algorithms is not efficient.

A prominent question related to the girth is finding exactly how dense a graph of given girth can be: the maximum number of edges in a d-regular graph with girth k is bounded by the well known *Moore bound* [2], which Alon later proved to be tight on general graphs as well [1]. Erdős conjectured that there exists a graph with $\Omega(n^{1+1/k})$ edges and girth $2k+1$ [12]. On the other hand, some have focused on giving practical lower bounds, i.e., finding ways to generate graphs of given girth as dense as possible [3,11]. We remark that our proposed algorithm `EBG-S` can match theory and practice: the densest n-vertex graph of girth k can be found as a subgraph of the complete graph K_n. While this may not be practical for large values of n, it significantly improves upon the brute force approach by avoiding the generation of subgraphs with girth <k.

2 Preliminaries

Let $G = (V(G), E(G))$ be a simple undirected graph with no self-loops, with vertex set $V(G)$ and edge set $E(G) \subseteq V(G) \times V(G)$. Two vertices u and v are *adjacent* (or neighbors) if there is an edge $e = \{u, v\} \in E(G)$ joining them. We

[1] The implementation of `EBG-S` in the github repository: https://github.com/ikn-lab/EnumerationAlgorithms/tree/master/BoundedGirth/.

[2] We remark that the techniques in [6] do not extend to undirected graphs, thus motivating a separate study. In directed graphs, a u-v path and a v-u path are distinct. However, a u-v path and a v-u path may be same in undirected graphs.

call e incident to v and we denote the set of incident edges to v $E(v)$. The set of neighbors of u in G is called its *neighborhood* and denoted by $N_G(u)$ and the size of $N_G(u)$ is called the *degree* of u in G. Let $N_G[u] = N_G(u) \cup \{u\}$ be the closed neighborhood of u. The *set of neighbors* of $U \subseteq V$ is defined as $N_G(U) = \bigcup_{u \in U} N_G(u) \setminus U$. Similarly, $N_G[U]$ denotes $N_G(U) \cup U$. For any vertex subset $S \subseteq V$, we call $G[S] = (S, E[S])$ an *induced subgraph*, where $E[S] = E(G) \cap (S \times S)$. Since $G[S]$ is uniquely determined by S, we sometimes identify $G[S]$ with S. For any edge subset $E' \subseteq E$, we call $G[E'] = (V'(E'), E')$ *edge induced subgraph*, where $V'(E') = \bigcup_{\{u,v\} \in E'} u$. We define $G \setminus \{e\} = (V, E \setminus \{e\})$ and $G \setminus \{v\} = G[V \setminus \{v\}]$. For simplicity, we use $v \in G$ and $e \in G$ to refer to $v \in V(G)$ and $e \in E(G)$, respectively. If G is clear from the context, we will also use simplified notation such as V, E, $N(u)$ instead of $V(G)$, $E(G)$, $N_G(u)$.

A sequence $P = (v_1, \ldots, v_{k+1})$ of distinct vertices is a *path* from v_1 to v_{k+1} (v_1-v_{k+1} path for short) in $G = (V, E)$ if for any $i \in [1, k]$, $\{v_i, v_{i+1}\} \in E$. P is a *shortest path* between two vertices if there is no shorter path between them. Let us denote by $V(P)$ and $E(P)$ the set of vertices and edges in P, respectively. We say that G is *connected* if for any two vertices $u, v \in V$, there is a u-v path. We say that a sequence $C = (v_1, \ldots, v_{k+1})$ of vertices is a *cycle* if $(v_1, \ldots, v_k)$ is a v_1-v_k path, $v_{k+1} = v_1$, and $\{v_k, v_{k+1}\} \in E$. The *length* of a path or cycle is defined by its number of edges. The *distance* between two vertices is the length of a shortest path between them. The *girth* of G, denoted by $g(G)$, is the length of a shortest cycle in G. For simplicity, we say that G has girth k if $g(G) \geq k$. The girth of acyclic graphs is usually assumed to be ∞.

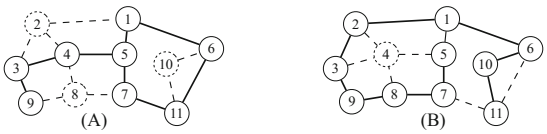


Fig. 1. Dashed edges and vertices are not included by an induced subgraph and a subgraph. An induced subgraph of girth five (A) and a subgraph of girth six (B).

We define our problems as follows and Fig. 1 shows examples of solutions Problem 1 and Problem 2. If we store all outputs, then it is easy to avoid duplicates. Our algorithms achieve without duplicates in polynomial space.

Problem 1 (k-girth connected induced subgraph enumeration). Enumerate all connected induced subgraphs S of a graph G with $g(S) \geq k$, without duplicates.

Problem 2 (k-girth connected subgraph enumeration). Enumerate all connected subgraphs S of a graph G with $g(S) \geq k$, without duplicates.

3 Enumeration by Binary Partition

The *binary partition method* is one of the fundamental frameworks for designing enumeration algorithms. Typically, a binary partition algorithm $\mathcal{A}$ has the fol-

Algorithm 1. Enumerate all connected induced subgraphs with girth k.

```
1 Procedure EBG(G, k)                    // G: an input graph, k: positive integer
2 |  RecEBG(∅, G);
3 Procedure RecEBG(S, G)                                  // S: the current solution
4 |  Output S;
5 |  DONE ← ∅;
6 |  for v ∈ C(S) do
7 |  |  RecEBG(S ∪ {v}, G \ DONE);
8 |  |  DONE ← DONE ∪ {v};
9 |  return;
```

lowing structure: first $\mathcal{A}$ picks an element x of the input, then divides the search space into two disjoint spaces, one containing the solutions that include x, and one those that do not. $\mathcal{A}$ recursively executes the above step until all elements are picked. Whenever the search space contains exactly one solution, $\mathcal{A}$ outputs it. We call each dividing step an *iteration*.

Algorithm EBG, detailed in Algorithm 1, represents a basic strategy for Problem 1. Algorithm 1 is based on binary partition, although each iteration divides the search space in more than two subspaces. While EBG enumerates solutions by picking vertices on each iteration, we can obtain an enumeration algorithm for Problem 2 by modifying EBG so that it picks edges instead.

Let G, X, and $S(X)$ be respectively an input graph, an iteration, and the solution received by the iteration X. A vertex $v \notin S(X)$ is a *candidate vertex* for $S(X)$ if $g(S(X) \cup \{v\}) \geq k$ and $S(X) \cup \{v\}$ is connected, that is, the addition of a candidate vertex generates a new solution. Let $C(S(X))$ be a set of candidate vertices for $S(X)$. We call $C(S(X))$ the *candidate set* of $S(X)$. Now, suppose that X generates new iterations $Y_1, \ldots, Y_d$ by adding vertices in $C(S(X)) = \{v_1, \ldots, v_d\}$ on line 7. For each i, we say that X is the *parent* of Y_i, and Y_i is a *child* of X. Note that, on iteration Y_i and its descendant iterations, EBG outputs solutions that do not include $v_1, \ldots, v_{i-1}$ but do include v_i. This implies that the solution space of Y_i is disjoint from those of each $Y_{j<i}$ created so far, i.e., EBG divides the solution space of X in d disjoint subspaces. The only iteration without a parent is the one generated on line 2, which we call the *initial iteration* and denote by I. We remark that $S(I) = \emptyset$ and that $\emptyset$ is a solution.

By using the above parent-child relation, we introduce the *enumeration tree* $\mathcal{T}(G) = \mathcal{T} = (\mathcal{V}, \mathcal{E})$. Here, $\mathcal{V}$ is the set of iterations of EBG for G and $\mathcal{E}$ is a subset of $\mathcal{V} \times \mathcal{V}$. For any pair of iterations X and Y, $(X, Y) \in \mathcal{E}$ if and only if X is the parent of Y. We can observe that $\mathcal{T}$ has no cycles since every child iteration of X receives a solution whose size is larger than $S(X)$. In addition, each iteration other than the initial iteration has exactly one parent. This implies that the initial iteration is an ancestor of all iterations and thus $\mathcal{T}$ is connected. Thus, $\mathcal{T}$ forms a tree. Next three lemmas show the correctness of EBG. Due to the space limitation, we omit some proofs (which can be found in Appendix).

Lemma 1. *Let G be a simple undirected graph and k a positive integer. Then, every output of* EBG *induces a connected subgraph of girth k.*

Lemma 2. *If X and Y are two distinct iterations on* EBG*, then $S(X) \neq S(Y)$.*

Lemma 3. *Let G be a simple undirected graph and k a positive integer.* EBG(G, k) *outputs all connected induced subgraphs with girth k in G exactly once.*

Proof. By Lemma 1, EBG outputs only solutions, and by Lemma 2 it does not output each solution more than once. We show that EBG outputs all solutions by induction. Let S be a solution. If $|S| = 0$, EBG outputs the empty set.

Otherwise, there is an iteration X_0 such that $S(X_0) \subseteq S$ and $S \subseteq V(G)$ (that is, no vertex of S has been removed from G). This is trivially true, e.g. for $X_0 = I$, since $S(I) = \emptyset$ and nothing has been removed from G. Note that every subgraph of a graph with girth at least k must also have girth at least k, thus every $v \in S \setminus S(X_0)$ such that $G[S(X_0) \cup \{v\}]$ is connected must be in $C\,(S(X_0))$. As S is connected there is at least one such v in $C\,(S(X_0))$.

Consider the first execution of Line 7 in X for which a vertex $v \in S \setminus S(X_0)$ is considered to generate a child iteration X_1. As no vertex of S was added to DONE in X_0, we still have that $S(X_1) \subseteq S$ and $S \subseteq V(G)$ in iteration X_1, but $|S(X_1)| = |S(X_0)| + 1$. Hence, by induction, EBG will eventually find S. □

Using Itai's algorithm [8] to compute the girth of a graph in $O(mn)$, we can obtain a first trivial complexity bound for Algorithm 1.

Theorem 1. EBG *solves Problem 1 with delay $O(n^2 m)$.*

Non-induced, weighted, and non-connected case. Let us briefly show how EBG also applies to some variants of the problem. Firstly, we can solve Problem 2, i.e., enumerate *edge* subgraphs, by modifying EBG as follows: Each solution is a set of edges $S \subseteq E$, and the candidate set $C\,(S(X))$ becomes $C\,(S(X)) = \{e \in E(X) \mid G[S(X) \cup \{e\}]$ is connected and $g(G[S(X) \cup \{v\}]) \geq k\}$. It is straightforward to see that Lemma 3 still holds (replacing the word *induced* with *edge* in the statement), and that the modified algorithm will solve Problem 2 in polynomial delay and polynomial space.

Furthermore, we can consider the *weighted* version of the problem, where the length of a cycle is the sum of the weights of its edges: we can find the girth in this case by adapting the Floyd-Warshall algorithm, and thus still enumerate all solutions for both the induced and edge subgraph version of the problem, in polynomial delay and polynomial space.

Finally, we consider non-connected case, i.e., where the solutions are all induced or edge subgraphs of girth k, and not just the connected ones: this is trivially done by redefining the candidate set as $C\,(S(X)) = \{v \in V(G) \mid g(G[S(X) \cup \{v\}]) \geq k\}$ for Problem 1, and similarly for Problem 2. If $G[S]$ is not connected, its girth is the minimum among that of its connected components, thus we can still use Itai's algorithm (or Floyd-Warshall if weighted edges are considered as well), and again obtain polynomial delay and polynomial space.

Algorithm 2. Updating data structures in `EBG-IS`.

Procedure `NextC`$(v, C(S), D^{(1)}(S), D^{(2)}(S), S, k, G)$
  $C(S \cup \{v\}) \leftarrow$ `UpdateCand`(v, S);
  $D^{(1)}(S \cup \{v\}) \leftarrow$ `Update1`$(v, C(S \cup \{v\}))$;
  $D^{(2)}(S \cup \{v\}) \leftarrow$ `Update2`$(v, C(S \cup \{v\}))$;
Function `UpdateCand`(v, S)
  $C(S \cup \{v\}) \leftarrow N(v) \cup C(S)$;
  foreach $u \in C(S)$ **do**
    if $D^{(1)}_{uv}(S) + D^{(2)}_{uv}(S) \geq k$ **then** $C(S \cup \{v\}) \leftarrow C(S \cup \{v\}) \cup \{u\}$;
  return $C(S \cup \{v\})$;
Function `Update1`$(v, C(S \cup \{v\}))$
  foreach $u \in C(S \cup \{v\}) \cup S, w \in C(S \cup \{v\})$ **do**
    $D^{(1)}_{uw}(S) \leftarrow \min\{D^{(1)}_{uw}(S), D^{(1)}_{uvw}(S)\}$
  return $D^{(1)}(S \cup \{v\})$
Function `Update2`$(v, C(S \cup \{v\}))$
  foreach $u, w \in C(S \cup \{v\})$ **do**
    $p_1 \leftarrow \min\{D^{(1)}_{uw}(S), D^{(1)}_{uvw}(S \cup \{v\}), D^{(2)}_{uw}(S)\}$;
    $p_2 \leftarrow$ the second smallest length in $\{D^{(1)}_{uw}(S), D^{(1)}_{uvw}(S \cup \{v\}), D^{(2)}_{uw}(S)\}$;
    if $p_1 + p_2 \geq k$ **then** // $x \in N(u) \cap S \cup \{v\}$
      $p_2 \leftarrow$ the second smallest length in $\{D^{(1)}_{xw}(S \cup \{v\}) + 1\}$;
    $D^{(2)}_{uw}(S \cup \{v\}) \leftarrow p_2$;
  return $D^{(2)}(S \cup \{v\})$;

4 Induced Subgraph Enumeration

The bottleneck of `EBG` is the computation of the candidate set. In this section, we present a more efficient algorithm `EBG-IS` for Problem 1. `EBG-IS` is based on `EBG`, but each iteration exploits information from the parent iteration, and maintains distances in order to improve the computation of the candidate set. The procedure is shown in Algorithm 2.

`EBG-IS` uses the second distance between vertices defined as follows. Let v be a vertex in $C(S) \cup S$, and u and u' be vertices in $C(S)$. We denote by $D^{(1)}_{uv}(S)$ the distance between v and u in $G[S \cup \{v, u\}]$, and by $D^{(2)}_{uu'}(S)$ the distance between u and u' in $G[S \cup \{u, u'\}] \setminus \{e_0\}$, where $e_0 = (u, \cdot)$ is the first edge on a shortest path between u and u'. Note that for any vertices $x \in G \setminus \{C(S) \cup S\}$, $y \in G \setminus C(S)$, and $y' \in G \setminus C(S)$, $D^{(1)}_{xy}(S) = \infty$ and $D^{(2)}_{yy'}(S) = \infty$. Especially, we call $D^{(2)}_{uu'}(S)$ the *second distance* between u and u' in $G[S \cup \{u, u'\}]$. In addition, we call a path whose length is the second distance a *second shortest path*. Moreover, we write $D^{(1)}_{uwv}(S)$ and $D^{(2)}_{uwv}(S)$ for the distance and the second distance from u to v via a vertex w, respectively. Let P and P' be respectively a v-u shortest path and a v-u second shortest path. Since P and P' do not share e_0 but do share their ends, H must have a cycle including v and u, where H is a subgraph of G such that $V(H) = V(P) \cup V(P')$ and $E(H) = E(P) \cup E(P')$. Figure 2(C) shows an

example of a cycle made by P and P'. To compute the candidate set efficiently, we will use the following lemmas. In the following lemmas, let X and Y be two iterations such that X is the parent of Y, and v be a vertex in $C(S(X))$ such that $S(Y) = S(X) \cup \{v\}$.

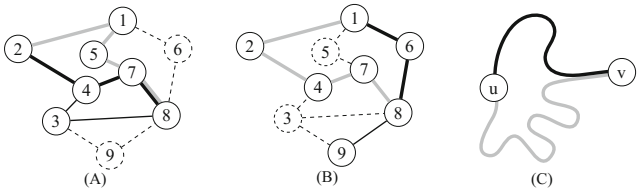


Fig. 2. (A) and (B) show two induced subgraphs. (C) shows a shortest path and a second shortest path. Dashed edges and vertices are not contained by induced subgraphs. Black and gray paths show respectively shortest and second shortest paths.

Lemma 4. *Let u and w be two vertices in $C(S(X))$ and $k = g(G[S(X)])$. (A) $g(G[S(X) \cup \{u, w\}]) \geq k$ if and only if (B) $D^{(1)}_{uw}(S(X)) + D^{(2)}_{uw}(S(X)) \geq k$.*

Proof. Clearly, (A) $\rightarrow$ (B) holds by definition of $D^{(1)}(S(X))$ and $D^{(2)}(S(X))$. For the direction (B) $\rightarrow$ (A), consider a shortest cycle C in $G[S(X) \cup \{u, w\}])$ in the following three cases: (I) $u, w \notin C$: $|C| \geq k$ since $g(G[S(X)]) \geq k$. (II) Either u or w in C: $|C| \geq k$ since u and w belong to $C(S(X))$. (III) Both u and w in C: C can be decomposed into two u-w paths P and Q. Without loss of generality, $|P| \leq |Q|$. If P is a u-w shortest path, then $|C| \geq k$ from (B), since Q is at least as long as the *second distance* $D^{(2)}_{uw}(S(X))$. Otherwise, there is a u-w shortest path P' and a cycle C' consisting of a part of P (or Q) and a part of P'. If C' contains w, then $|C'| = |C| \geq k$ since C is a shortest cycle. If C' does not contain w, then $|C'|$ is a cycle in $G[S(X) \cup \{u\}]$, thus $|C'| \geq k$ because $u \in C(S(X))$. □

Lemma 5. `EBG-IS` *computes $C(S(Y))$ in $O(|C(S(X))| + |N(v)|)$ time.*

Proof. From Lemma 4, vertex u in $C(S(X))$ belongs to $C(S(Y))$ if and only if $D^{(1)}_{uv}(S(X)) + D^{(2)}_{uv}(S(X)) \geq k$. This can be done in constant time. In addition, from the connectivity of $G[S(Y)]$, $C(S(Y)) \setminus C(S(X)) \subseteq N(v)$. Thus, we can find $C(S(Y)) \setminus C(S(X))$ in $O(|C(S(X))| + |N(v)|)$ time. □

Next, we consider how to update the values of $D^{(1)}(S(Y))$ and $D^{(2)}(S(Y))$ when adding v to $S(X)$. We can update the old distances to the ones after adding v as in the Floyd-Warshall algorithm (see Algorithm 2), meaning that we can compute $D^{(1)}(S(Y))$ in $O(|S(X) \cup C(S(X))| \cdot |C(S(X))|)$ time. By the following lemma, the values of $D^{(2)}(S(Y))$ can be updated in $O(|S(Y)|)$ time for each pair of vertices in $C(S(Y))$.

Lemma 6. *Let u and w be two vertices in $C(S(X))$, e_0 be an edge in a u-w shortest path in $G[S(X) \cup \{u, w\}]$, and $H = G[S(X) \cup \{u, w\}] \setminus \{e_0\}$. If $N_H(u) = \emptyset$, then $D^{(2)}_{uw}(S(X)) = \infty$. Otherwise, $D^{(2)}_{uw}(S(X)) = \min_{y \in N_H(u)}\{D^{(1)}_{yw}(S(X)) + 1\}$.*

Proof. From the definition of $D^{(2)}_{uw}(S(X))$, if $N_H(u) = \emptyset$, then $D^{(2)}_{uw}(S(X)) = \infty$. We assume $|N_H(u)| \geq 1$. Since $u \notin S(X)$, every shortest path between u and w in $G[S(X) \cup \{w\}] \cup f$ contains f, where $f = \{u, y\}$. Hence, $D^{(1)}_{yw}(S(X)) + 1$ is equal to the distance between u and w in $G[S(X) \cup \{w\}] \cup \{f\}$. Hence, the statement holds. □

The next lemma implies that if $D^{(1)}_{uw}(S(X)) + D^{(2)}_{uw}(S(X)) < k$, i.e., $G[S(X) \cup \{u, w\}]$ is not a solution, then computing $D^{(2)}_{uw}(S(Y))$ takes constant time.

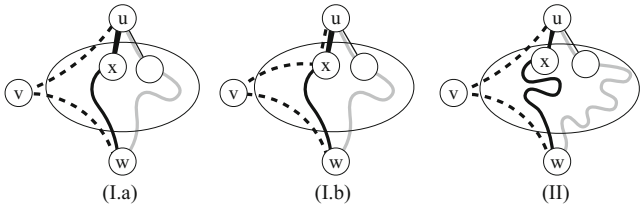


Fig. 3. Examples of each case in Lemma 7. Solid lines are u-v shortest paths in $G[S(X) \cup \{u, w\}]$. Gray solid lines are u-v second shortest paths in $G[S(X) \cup \{u, w\}]$. Dashed lines are u-v-w shortest paths in $G[S(Y) \cup \{u, w\}]$. Let $\{u, x\}$ be the first edge in a shortest path: the sum of lengths of a solid and gray solid line is less than k.

Lemma 7. *Let u and w be two vertices in $C(S(Y))$. If $p_1 + p_3 < k$, then $D^{(2)}_{uw}(S(Y)) = \min\{\max\{p_1, p_2\}, p_3\}$, where $p_1 = D^{(1)}_{uw}(S(X))$, $p_2 = D^{(1)}_{uvw}(S(Y))$, and $p_3 = D^{(2)}_{uw}(S(X))$.*

Proof. Let $G_X = G[S(X) \cup \{u, w\}]$ and $G_Y = G[S(Y) \cup \{u, w\}]$. Note that $p_1 \leq p_3$. We consider the following cases: (I) $p_1 < p_2$: Let $e = \{u, x\}$ be the first edge of a u-w shortest path P in G_Y. Note that P cannot contain v. (I.a) There exists a u-v-w shortest path Q that does not contain e: clearly, $D^{(2)}_{uw}(S(Y)) = \min\{|Q| = p_2, p_3\}$. (I.b) Every u-v-w shortest path Q contains e: there always exists a cycle C in $S(Y) \cup \{w\}$ such that $V(C) \subseteq (V(P) \cup V(Q)) \setminus \{u\}$ and C does not contain u. Note that $|C| < p_1 + p_2$. If $p_2 \leq p_3$, then this contradicts $w \in C(S(Y))$ since $|C| < k$. Thus, $p_2 > p_3$. This implies that $|Q| - 1 \geq p_3$. Hence, $D^{(2)}_{uw}(S(Y)) = p_3$. (II) $p_2 \leq p_1$: this assumption implies that there exists a u-w shortest path P in G_Y that contains v, and $p_1 + p_2 < k$. Let e be the first edge of P in G_Y and Q be a u-v-w shortest path in $G_Y \setminus \{e\}$. Now, we can see $|Q| > p_1$ since if $|Q| \leq p_1$, then $u \notin C(S(Y))$ since P and Q make a cycle C containing u with $|C| < k$. Thus, the length of a u-w shortest path in $G_Y \setminus \{e\}$ is p_1, and $D^{(2)}_{uw}(S(Y)) = p_1$ holds. □

Algorithm 2 shows in detail the update of the candidate set, $D^{(1)}(\cdot)$, and $D^{(2)}(\cdot)$ (done using Lemma 7). We analyze the time complexity of EBG-IS. Let $ch(X)$ be the set of children of X and $\#gch(X)$ be the number of grandchildren of X. The next lemma shows the time complexity for updating $D^{(2)}(S(X))$.

Lemma 8. *We can compute* $D^{(2)}(S(Y))$ *from* $D^{(2)}(S(X))$ *in* $O(\#gch(Y) \cdot |S(Y)| + |C(S(Y))|^2)$ *time.*

Proof. Let u and w be two vertices in $C(S(Y))$. Two cases are possible: (I) $D^{(1)}_{uw}(S(X)) + D^{(2)}_{uw}(S(X)) \geq k$: By Lemma 6, computing $D^{(2)}_{uw}(S(Y))$ takes $O(|S(Y)|)$ time, checking only vertices in $S(Y)$. As the number of pairs (u, w) that fit this case is bounded by $\#gch(Y)$, EBG-IS needs $O(\#gch(Y) \cdot |S(Y)|)$ time to compute this part. (II) $D^{(1)}_{uw}(S(X)) + D^{(2)}_{uw}(S(X)) < k$: From Lemma 7, computing $D^{(2)}_{uw}(S(Y))$ takes constant time, for a total complexity of $O(|C(S(Y))|^2)$, which proves the statement. □

Theorem 2. EBG-IS *enumerates all solutions in* $O(\sum_{S \in \mathcal{S}} |N[S]|)$ *time using* $O(\max_{S \in \mathcal{S}}\{|N[S]|^3\})$ *space, where* $\mathcal{S}$ *is the set of all solutions.*

Proof. The correctness of EBG-IS follows from Lemma 3. We first consider the space complexity. In an iteration X, EBG-IS uses $O(|C(S(X)) \cup S(X)|^2)$ space for storing values of $D^{(1)}(\cdot)$ and $D^{(2)}(\cdot)$. In addition, the height of $\mathcal{T}$ is at most $\max_{S \in \mathcal{S}}\{|S|\}$. Therefore, EBG-IS uses $O(\max_{S \in \mathcal{S}}\{|N[S]|^3\})$ space.

Let $c(X)$ be $|C(S(X))|$ and $T(X, Y)$ be the time needed to generate Y from X, i.e., an execution of NextC() (Algorithm 2). From Lemma 5, Lemma 6, and the Floyd-Warshall algorithm, $T(X, Y)$ is $O(c(X) + |N(v)| + c(Y) \cdot |S(X)| + \#gch(Y) \cdot |S(Y)| + c(Y)^2)$ time. In addition, $|N[S(X)]| \leq |N[S(Y)]|$, $|N(v)| = O(|N[S(Y)]|)$, and $c(X) = O(N[S(X)])$ since every vertex in the candidate set has a neighbor in $S(X)$. Thus, $T(X, Y) = O(|N[S(Y)]|\,(c(Y) + \#gch(Y)))$ time. Note that the sum of children and grandchildren for all iterations is at most $2|\mathcal{V}|$. Thus, by distributing the $O(|N[S(Y)]|)$ time from X to children and grandchildren of Y, each iteration needs $O(|N[S(Y)]|)$ time since each iteration receives costs only from the parent and the grandparent. In addition, each iteration outputs a solution, and hence the total time is $O(\sum_{S \in \mathcal{S}} |N[S]|)$. □

5 Subgraph Enumeration

We propose an algorithm, EBG-S, for enumerating all subgraphs with girth k in a given graph G, detailed in Algorithm 3. A trivial adaptation of EBG-IS would run in $O(m)$ time per solution, as the candidate sets are sets of edges, whose size is $O(m)$. To improve this running time, EBG-S selects candidates in a certain order, so that the number of candidate edges does not exceed no more than the number of nodes in the previous solution $G[S]$.

Let S be the current solution. Note that S is an edge set. We first define an inner edge and an outer edge as follows: an edge $e = \{u, v\}$ is an *inner edge*

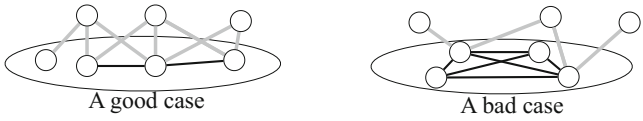


Fig. 4. Black solid lines and gray solid lines represent inner edges and outer edges, respectively. Our main strategy is to reduce the number of inner edges in `EBG-S`.

for S if $u, v \in G[S]$, and an *outer edge* otherwise (see Fig. 4). Let $C_{\mathrm{in}}(S)$ and $C_{\mathrm{out}}(S)$ be a set of inner edges and outer edges in $C(S)$, respectively. We first consider the case when `EBG-S` picks an outer edge. In the following lemmas, let X be an iteration in enumeration tree $\mathcal{T}$, e be an edge not in X, and Y be the child iteration of X satisfying $S(Y) = S(X) \cup \{e\}$.

Lemma 9. *Let $e = \{x, y\}$ be an outer edge such that $x \in V(G[S(X)])$. Then $C(S(Y)) \subseteq (C(S(X)) \cup E(y)) \setminus \{e\}$, where $E(y)$ are the edges incident to y.*

Proof. An edge $g \notin E(y) \cup C(S(X))$ may not be added to $S(Y)$ as the resulting subgraph would be disconnected, and $e \notin C(S(Y))$ since $e \in S(Y)$. □

From Lemma 9, `EBG-S` manages the candidate set $C(S(Y))$ in $O(|C(S(Y))| + |V(G[S(X)])|)$ time when `EBG-S` picks an outer edge e since we can add all edges $e' \notin S(X) \cup C(S(X))$ incident to y and $S(Y) \cup \{e'\}$ is a solution. Moreover, removed edges are at most $|V(G[S(X)])|$ since all removed edges have a vertex in $V(G[S(X)])$. In this case, `EBG-S` can obtain $C_{\mathrm{in}}(S(Y))$ and $C_{\mathrm{out}}(S(Y))$ in $O(S(X))$ time and $O(C(S(Y)))$ time, respectively. Next, we consider that when `EBG-S` picks an inner edge e. When we pick an inner edge, $C(S(Y))$ is monotonically decreasing.

Lemma 10. *If e is an inner edge, then $C_{\mathrm{in}}(S(Y)) \subset C_{\mathrm{in}}(S(X))$ and $C_{\mathrm{out}}(S(Y)) = C_{\mathrm{out}}(S(X))$.*

Proof. Since e is an inner edge $V(G[S(Y)]) = V(G[S(X)])$, thus there is no edge $f \in C_{\mathrm{in}}(S(Y)) \setminus C_{\mathrm{in}}(S(X))$. Since $e \notin C_{\mathrm{in}}(S(Y))$ and no edge in $C_{\mathrm{out}}(S(X))$ is in $C_{\mathrm{in}}(S(Y))$, $C_{\mathrm{in}}(S(Y)) \subset C_{\mathrm{in}}(S(X))$. Moreover, there is no cycle including $f \in C_{\mathrm{out}}(S(X))$ in $G[S(Y) \cup \{f\}]$, hence $C_{\mathrm{out}}(S(Y)) = C_{\mathrm{out}}(S(X))$. □

Next, for any pair of edges e and f not in $G[S(X)]$, we consider the computation of the girth of $G[S(X) \cup \{e, f\}]$ in `EBG-S`. Let $A(X) = \{v \in V(G[S(X)]) \mid E(v) \cap C(S(X)) \neq \emptyset\}$. In a similar fashion as `EBG-IS`, `EBG-S` uses $D^{(3)}(S(X))$ for $A(X)$. The definition of $D^{(3)}(S(X))$ is as follows: For any pair of vertices u and v in $A(X)$, $D^{(3)}_{uv}(S(X))$ is the distance between u and v in $A(X)$. Note that a shortest path between u and v may contain a vertex in $G[S] \setminus A(X)$. The next lemma shows that by using $D^{(3)}(S(X))$, we can compute $C(S(Y))$ in $O(|V(G[S(Y)])|)$ time from $C(S(X))$.

Lemma 11. *For any iteration X, $|C_{\mathrm{in}}(S(X))| \leq |V(G[S(X)])|$.*

Algorithm 3. Updating data structures in `EBG-S`.

Procedure `NextC`$(C(S), D^{(3)}(S), S, k, G)$
  if $C_{\mathrm{in}}(S) \neq \emptyset$ **then** $e \leftarrow C_{\mathrm{in}}(S)$; **else** $e \leftarrow C_{\mathrm{out}}(S)$;
  $C(S \cup \{e\}) \leftarrow$ `UpdateCand`(e, S);
  $D^{(3)}(S \cup \{e\}) \leftarrow$ `Update3`$(e, C(S \cup \{e\}))$;
Function `UpdateCand`$(e = \{u, v\}, S)$
  if $e \in C_{\mathrm{in}}(S)$ **then**
    for $f \in C_{\mathrm{in}}(S) \setminus \{e\}$ **do**
      if $g(G[S \cup \{e, f\}]) \geq k$ **then** $C_{\mathrm{in}}(S) \leftarrow C_{\mathrm{in}}(S) \cup \{f\}$;
  else // We assume $u \in G[S]$ and $v \notin G[S]$
    for $w \in N(v)$ **do** // Let f be an edge $\{v, w\}$
      if $g(G[S \cup \{e, f\}]) < k$ **then** $C_{\mathrm{out}}(S) \leftarrow C_{\mathrm{out}}(S) \setminus f$;
      else if $w \in G[S]$ **then**
        $(C_{\mathrm{in}}(S), C_{\mathrm{out}}(S)) \leftarrow (C_{\mathrm{in}}(S) \cup f, C_{\mathrm{out}}(S) \setminus f)$
      else $C_{\mathrm{out}}(S) \leftarrow C_{\mathrm{out}}(S) \cup f$;
  return $C_{\mathrm{in}}(S) \cup C_{\mathrm{out}}(S)$;
Function `Update3`$(e = \{u, v\}, C(S \cup \{e\}))$
  $A = \{v \in V(G[S]) \mid v$ is incident to $C(S).\}$;
  for $x, y \in A$ **do** // If $e \in C_{\mathrm{out}}(S)$, then $u \in V(G[S]), v \notin V(G[S])$
    if $e \in C_{\mathrm{in}}(S)$ **then**
      $D^{(3)}_{xy}(S) \leftarrow \min\{D^{(3)}_{xy}(S), D^{(3)}_{xu}(S) + D^{(3)}_{vy}(S) + 1, D^{(3)}_{xv}(S) + D^{(3)}_{uy}(S) + 1\}$;
    else $D^{(3)}_{xy}(S) \leftarrow \min\{D^{(3)}_{xy}(S), D^{(3)}_{xu}(S) + 1\}$;
  return $D^{(3)}(S)$;

Proof. The proof follows from these facts: (A) Initially, $C_{\mathrm{in}}(S(X)) = \emptyset$. (B) Choosing $e \in C_{\mathrm{in}}(S(X))$ decreases $|C_{\mathrm{in}}(S(Y))|$. (C) $e = \{x, y\} \in C_{\mathrm{out}}(S(X))$ is chosen iff $|C_{\mathrm{in}}(S(X))| = 0$, and (assuming wlog $y \notin V(G[S(X)])$) it increases $|C_{\mathrm{in}}(S(Y))|$ by at most $|\{\{y, z\} : z \in V(G[S(X)])\}| < |V(G[S(X)])|$. □

Lemma 12. $|C_{\mathrm{out}}(S(X)) \setminus C_{\mathrm{out}}(S(Y))| + |C_{\mathrm{out}}(S(Y)) \setminus C_{\mathrm{out}}(S(X))| \leq V(G[S(Y)])$.

Proof. We consider two cases: (I) $C_{\mathrm{in}}(S(X)) \neq \emptyset$: `EBG-S` picks $e \in C_{\mathrm{in}}(S(X))$, and thus, From Lemma 10, $C_{\mathrm{out}}(S(Y)) = C_{\mathrm{out}}(S(X))$. (II) $C_{\mathrm{in}}(S(X)) = \emptyset$: `EBG-S` picks $e = \{u, v\} \in C_{\mathrm{out}}(S(X))$. Without loss of generality, we can assume that $u \in V(G[S(X)])$ and $v \notin V(G[S(X)])$. Let f be an edge $\{v, w\}$ incident to v. Now, $w \in V(G[S(Y)])$. This implies that the number of edges that are added to $C_{\mathrm{out}}(S(Y))$ and removed from $C_{\mathrm{out}}(S(X))$ is at most $|V(G[S(Y)])|$. □

Note that $|V(G[S(X)])| \leq |V(G[S(Y)])|$. Hence, from the above lemmas, we can obtain the following lemma.

Lemma 13. *$C(S(Y))$ can be computed in $O(|V(G[S(Y)])|)$ time from $C(S(X))$.*

Theorem 3. `EBG-S` *enumerates all connected subgraphs with girth k in $O(\sum_{S \in \mathcal{S}} |V(G[S])|)$ total time using $O(\max_{S \in \mathcal{S}}\{|V(G[S])|^3\})$ space.*

Proof. The proof can be obtained by adapting that of Theorem 2. A more detailed proof can be found in the appendix. □

6 Conclusion

In this paper, we addressed the k-girth connected induced/edge subgraph enumeration problems. We proposed two algorithms: `EBG-IS` for induced subgraphs and `EBG-S` for edge subgraphs. Both algorithms have $O(n)$ time delay and require $O(n^3)$ space (exact bounds are reported in Table 1). The algorithms can easily be adapted to relax the connectivity constraint and consider weighted graphs. Other possibilities include applying the algorithms for network analysis and considering the more challenging problem of enumerating maximal subgraphs.

Table 1. Summary of our result. $\mathcal{S}$ is the set of all solutions.

	Total time	Total space
`EBG-IS`	$O(\sum_{S\in\mathcal{S}} \|N[S]\|)$	$O(\max_{S\in\mathcal{S}}\{\|N[S]\|^3\})$
`EBG-S`	$O(\sum_{S\in\mathcal{S}} \|V(G[S])\|)$	$O(\max_{S\in\mathcal{S}}\{\|V(G[S])\|^3\})$

References

1. Alon, N., Hoory, S., Linial, N.: The moore bound for irregular graphs. Gr. Comb. **18**(1), 53–57 (2002)
2. Bollobás, B.: Extremal Graph Theory. Courier Corporation (2004)
3. Chandran, L.S.: A high girth graph construction. SIAM J. Discrete Math. **16**(3), 366–370 (2003)
4. Chang, H.-C., Lu, H.-I.: Computing the girth of a planar graph in linear time. SIAM J. Comput. **42**(3), 1077–1094 (2013)
5. Conte, A., Kanté, M.M., Otachi, Y., Uno, T., Wasa, K.: Efficient enumeration of maximal k-degenerate subgraphs in a chordal graph. In: Cao, Y., Chen, J. (eds.) COCOON 2017. LNCS, vol. 10392, pp. 150–161. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62389-4_13
6. Conte, A., Kurita, K., Wasa, K., Uno, T.: Listing acyclic subgraphs and subgraphs of bounded girth in directed graphs. In: Gao, X., Du, H., Han, M. (eds.) COCOA 2017. LNCS, vol. 10628, pp. 169–181. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-71147-8_12
7. Ferreira, R., Grossi, R., Rizzi, R.: Output-sensitive listing of bounded-size trees in undirected graphs. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 275–286. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23719-5_24
8. Itai, A., Rodeh, M.: Finding a minimum circuit in a graph. SIAM J. Comput. **7**(4), 413–423 (1978)
9. Johnson, D.S., Yannakakis, M., Papadimitriou, C.H.: On generating all maximal independent sets. Inf. Process. Lett. **27**(3), 119–123 (1988)

10. Kurita, K., Wasa, K., Arimura, H., Uno, T.: Efficient enumeration of dominating sets for sparse graphs. arXiv preprint arXiv:1802.07863 (2018)
11. 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)
12. Parter, M.: Bypassing Erdős' girth conjecture: hybrid stretch and sourcewise spanners. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8573, pp. 608–619. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43951-7_49
13. Read, R.C., Tarjan, R.E.: Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. Networks **3**(5), 237–252 (1975)
14. Shioura, A., Tamura, A., Uno, T.: An optimal algorithm for scanning all spanning trees of undirected graphs. SIAM J. Comput. **26**(3), 678–692 (1997)
15. Wasa, K., Arimura, H., Uno, T.: Efficient enumeration of induced subtrees in a K-degenerate graph. In: Ahn, H.-K., Shin, C.-S. (eds.) ISAAC 2014. LNCS, vol. 8889, pp. 94–102. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13075-0_8

An Optimal Algorithm for Online Prize-Collecting Node-Weighted Steiner Forest

Christine Markarian(✉)

Heinz Nixdorf Institute, Computer Science Department,
Paderborn University, Fürstenallee 11, 33102 Paderborn, Germany
chrissm@mail.uni-paderborn.de

Abstract. We study the *Online Prize-collecting Node-weighted Steiner Forest* problem (OPC-NWSF) in which we are given an undirected graph $G = (V, E)$ with $|V| = n$ and node-weight function $w : V \to \mathcal{R}^+$. A sequence of k pairs of nodes of G, each associated with a penalty, arrives online. OPC-NWSF asks to construct a subgraph H such that each pair $\{s, t\}$ is either connected (there is a path between s and t in H) or its associated penalty is paid. The goal is to minimize the weight of H and the total penalties paid. The current best result for OPC-NWSF is a randomized $\mathcal{O}(\log^4 n)$-competitive algorithm due to Hajiaghayi *et al.* (ICALP 2014). We improve this by proposing a randomized $\mathcal{O}(\log n \log k)$-competitive algorithm for OPC-NWSF, which is optimal up to constant factor since OPC-NWSF has a randomized lower bound of $\Omega(\log^2 n)$ due to Korman [11]. Moreover, our result also implies an improvement for two special cases of OPC-NWSF, the *Online Prize-collecting Node-weighted Steiner Tree* problem (OPC-NWST) and the *Online Node-weighted Steiner Forest* problem (ONWSF). In OPC-NWST, there is a distinguished node which is one of the nodes in each pair. In ONWSF, all penalties are set to infinity. The currently best known results for OPC-NWST and ONWSF are a randomized $\mathcal{O}(\log^3 n)$-competitive algorithm due to Hajiaghayi *et al.* (ICALP 2014) and a randomized $\mathcal{O}(\log n \log^2 k)$-competitive algorithm due to Hajiaghayi *et al.* (FOCS 2013), respectively.

Keywords: Online algorithms · Competitive analysis · Steiner forest
Steiner tree · Prize-collecting · Node-weighted graphs · Penalties

1 Introduction

Steiner problems, which ask for a minimum weight subgraph of a given (undirected) graph that satisfies some connectivity requirements, have been of significant interest over the past few decades. They have been originally studied in

This work was partially supported by the Federal Ministry of Education and Research (BMBF) as part of the project 'Resilience by Spontaneous Volunteers Networks for Coping with Emergencies and Disaster' (RESIBES), (grant no. 13N13955 to 13N13957).

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 214–223, 2018.
https://doi.org/10.1007/978-3-319-94667-2_18

edge-weighted graphs and later in node-weighted graphs. Node-weighted variants generalize edge-weighted ones by replacing each edge by a node with the corresponding edge weight. In this paper, we are interested in node-weighted graphs. A broader class of Steiner problems in which violation of some of the requirements is possible at the cost of paying an associated penalty, is known as *prize-collecting* Steiner problems. These were motivated by client-server network planning scenarios in which a service provider may decide to lose some of his clients or refuse to take some new ones, thereby paying an associated penalty cost. In many scenarios, connectivity requirements are not known in advance but are revealed over time (arrival of new clients). These are modeled as *online* Steiner problems in which the algorithm has to satisfy a connectivity requirement as soon as revealed by irrevocably making immediate decisions. In this paper, we study the *Online Prize-collecting Node-weighted Steiner Forest* problem (OPC-NWSF), introduced by Hajiaghayi *et al.* [8] and defined as follows.

Definition 1 *(OPC-NWSF). Given an undirected graph $G = (V, E)$ with $|V| = n$ and node-weight function $w : V \to \mathcal{R}^+$. A sequence of k pairs of nodes of G (called terminal pairs), each associated with a penalty, arrives online. OPC-NWSF asks to construct a subgraph H such that each pair $\{s, t\}$ is either connected (there is a path between s and t in H) or its associated penalty is paid. The goal is to minimize the weight of H and the total penalties paid.*

Throughout the paper, unless stated otherwise, we represent the number of nodes in the input graph by n and the number of terminal pairs by k. We assume the standard model for online Steiner problems in which the input graph is known in advance while the terminal pairs arrive over time. We measure the performance of the online algorithm using the classical notion of *competitive analysis*. Given an input sequence σ, let $\mathcal{C}_A(\sigma)$ and $\mathcal{C}_{OPT}(\sigma)$ denote the cost incurred by an algorithm A and an optimal offline algorithm OPT, respectively. Algorithm A is said to be *c-competitive* (or has competitive ratio c) if there exists a constant α such that $\mathcal{C}_A(\sigma) \leq c \cdot \mathcal{C}_{OPT}(\sigma) + \alpha$ for all input sequences σ. OPC-NWSF admits a randomized $\mathcal{O}(\log^4 n)$-competitive algorithm due to Hajiaghayi *et al.* [8], based on a generic technique that reduces online prize-collecting Steiner problems to their corresponding fractional non-prize-collecting variants, by losing a logarithmic factor in the competitive ratio. OPC-NWSF generalizes the *Online Set Cover* problem (OSC) introduced by Alon *et al.* [1] and for which Korman [11] has given a randomized lower bound of $\mathcal{O}(\log n \log m)$ on its competitive ratio, where n denotes the number of elements and m denotes the number of subsets. A randomized lower bound on the competitive ratio for OPC-NWSF is thus $\mathcal{O}(\log^2 n)$.

Our Contribution. We propose an online polynomial-time $\mathcal{O}(\log n \log k)$ competitive randomized algorithm for OPC-NWSF, which is optimal up to constant factor. Our algorithm can be viewed as an online LP rounding algorithm and draws on ideas from Alon *et al.*'s algorithm for the *Generalized Connectivity* problem (GN) [2] in which the given graph is an edge-weighted graph and a

sequence of pairs of nodes arrives online. GN asks to construct a minimum-weight subgraph which contains a path between each given pair. Our result:

- improves the previous best result for OPC-NWSF, the randomized $\mathcal{O}(\log^4 n)$-competitive algorithm due to Hajiaghayi *et al.* (Theorem 3 in [8] combined with Theorem 1 in [7]).
- improves the previous best result and gives an optimal algorithm for a special case of OPC-NWSF, the *Online Prize-collecting Node-weighted Steiner Tree* problem (OPC-NWST) in which there is a distinguished node that is one of the nodes in each pair. The previous best result for OPC-NWST is a randomized $\mathcal{O}(\log^3 n)$-competitive algorithm due to Hajiaghayi *et al.* (Corollary 1 in [8]).
- improves the previous best result and gives an optimal algorithm for a special case of OPC-NWSF, the *Online Node-weighted Steiner Forest* problem (ONWSF) in which all penalties are set to infinity. The previous best result for ONWSF is a randomized $\mathcal{O}(\log n \log^2 k)$-competitive algorithm due to Hajiaghayi *et al.* (Corollary 1 in [7]). Note that the *Generalized Connectivity* problem by Alon *et al.* [2] can be seen as the edge-weighted counterpart of ONWSF in which edges rather than nodes have weights.
- matches the optimal competitive ratio due to Hajiaghayi *et al.* (Theorem 2 in [8]) for a special case of OPC-NWSF, the *Online Node-weighted Steiner Tree* problem (ONWST) in which there is a distinguished node that is one of the nodes in each pair *and* all penalties are set to infinity.

Outline. The rest of the paper is organized as follows. In Sect. 2, we give an overview of related literature. In Sect. 3, we present our randomized algorithm and show its analysis in Sect. 4. We conclude in Sect. 5 with some future work.

2 Related Work

Online Steiner problems were initially studied in edge-weighted graphs. Imase and Waxman [9] have shown that a natural greedy algorithm achieves an $\mathcal{O}(\log n)$-competitive ratio for the *Online Steiner Tree* problem (OST) in edge-weighted graphs, which is optimal up to constant factor. Awerbuch *et al.* [4] have generalized this result by showing that the greedy algorithm achieves an $\mathcal{O}(\log^2 n)$-competitive ratio for the *Online Steiner Forest* problem (OSF) in edge-weighted graphs. Berman and Coulston [5] have improved the latter to an $\mathcal{O}(\log n)$-competitive ratio. The *Online Prize-collecting Steiner Tree* problem (OPST) in edge-weighted graphs was studied by Qian and Williamson [14] who have given an $\mathcal{O}(\log n)$-competitive algorithm, based on an online primal-dual approach. Hajiahjayi *et al.* [8] have given the same competitive ratio for OPST and an $\mathcal{O}(\log n)$-competitive algorithm for the *Online Prize-collecting Steiner Forest* problem (OPST) in planar edge-weighted graphs. Naor *et al.* [13] have initiated the study of online Steiner problems in node-weighted graphs by giving an $\mathcal{O}(\log^3 n)$-competitive randomized algorithm for the *Online Node-weighted*

Steiner Tree problem (ONWST). Their algorithm is based on *spider decomposition*, a technique first introduced by Klein and Ravi [10] for the offline version of the problem. Hajiahjayi *et al.* [8] have later improved this result to an $\mathcal{O}(\log^2 n)$-competitive ratio by employing a new technique for dual-fitting analysis called *dual averaging*. This ratio is optimal up to constant factor since ONWST generalizes the *Online Set Cover* problem (OSC) [1] which has a randomized lower bound of $\mathcal{O}(\log n \log m)$ [11], where n denotes the number of elements and m denotes the number of subsets. In fact, node-weighted Steiner problems strictly generalize their edge-weighted counterparts since every edge can be replaced by a node with the corresponding edge weight. Angelopoulos [3] has studied a variant of ONWST in which node-weights are restricted, such that the ratio of the maximum node-weight to the minimum node-weight is bounded (and appears in the competitive ratio). Hajiahjayi *et al.* [7] have proposed an $\mathcal{O}(\log^3 n)$-competitive randomized algorithm for the *Online Node-weighted Steiner Forest* problem (ONWSF). Their result is based on a technique called *disk paintings* in which primal cost updates are amortized to a set of mutually disjoint fixed-radius dual disks centered at a subset of terminals. Hajiahjayi *et al.* [8] have proposed a generic technique that reduces online prize-collecting Steiner problems to their corresponding fractional non-prize-collecting variants, by losing a logarithmic factor in the competitive ratio. This has implied $\mathcal{O}(\log^3 n)$-competitive and $\mathcal{O}(\log^4 n)$-competitive randomized algorithms for the *Online Prize-collecting Node-weighted Steiner Tree* problem (OPC-NWST) and the *Online Prize-collecting Node-weighted Steiner Forest* problem (OPC-NWSF), respectively. For OPC-NWSF in planar graphs, they have shown an $\mathcal{O}(\log^2 n)$-competitive ratio.

3 Online Algorithm

In this section, we present our randomized online algorithm for OPC-NWSF.

The algorithm is given an undirected graph $G = (V, E)$ with node-weight function $w : V \to \mathcal{R}^+$. In each step, a pair $\{s, t\}$ associated with a penalty p needs to be *served* by either paying its penalty or assuring a path between s and t in the subgraph induced by the set of nodes outputted by the algorithm.

The algorithm assigns a fraction, initially set to 0, to each of the n nodes of G *and* to each of the k penalties associated with the pairs. The *maximum flow* between nodes s and t in an undirected edge-weighted graph $G = (V, E)$ is defined as the smallest total weight of edges of G which if removed would disconnect s from t. These edges form a *minimum cut* between s and t in G. Alon *et al.*'s fractional algorithm [2] satisfies a connectivity requirement between two nodes in an undirected edge-weighted graph by constructing a minimum cut between the two nodes and augmenting the fraction of each edge in the cut, until the maximum flow between the two nodes becomes at least 1. Since the fraction of each edge on a path connecting the two nodes has a value of at least the maximum flow going through the path, a fractional solution is guaranteed.

Upon the arrival of a new pair, our algorithm constructs a fractional solution by transforming the given node-weighted graph into an edge-weighted graph and

then exploiting a similar approach as Alon *et al.*'s [2] for edge-weighted graphs. It performs the following *edge-weight update*:

- It assigns a (temporary) weight to each edge e and sets its value to the smaller among the two fractional values corresponding to e's endpoint nodes. If an edge has s or t as one of its endpoint nodes and u as the other, the algorithm sets its weight to that of u.
- It adds a (temporary) *virtual edge* from s to t and sets its weight to the fractional value corresponding to p (see Fig. 1). This edge is removed after serving pair $\{s, t\}$.

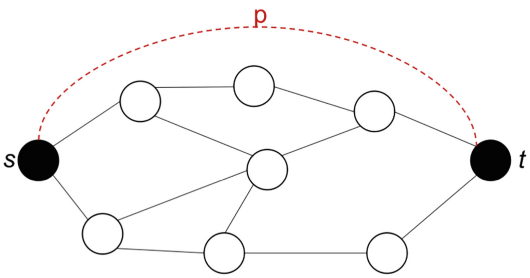


Fig. 1. Virtual edge of weight p from s to t

While the maximum flow between s and t is less than 1, the algorithm performs an edge-weight update and computes a minimum cut C between s and t by running the minimum cut polynomial-time algorithm by Schroeder *et al.* for undirected edge-weighted graphs [15]. Consider an edge e in C. We call the endpoint node of e with fractional value smaller than that of the other endpoint of e, a *minimum cut node* of C. The algorithm increases the fractions of all minimum cut nodes of C as well as the fraction corresponding to p. Note that the virtual edge from s to t belongs to every (minimum) cut between s and t. Since each edge e on any path from s to t has a weight of at least the maximum flow going through the path and this value is the smaller of the two fractional values corresponding to the two endpoint nodes, a fractional solution is guaranteed. The algorithm then rounds the fractional solution constructed. A variable μ is randomly chosen as the minimum among $2\lceil \log(k+1) \rceil$ independent random variables, distributed uniformly in the interval $[0, 1]$. The algorithm adds to the integral solution every node or penalty with fractional value exceeding μ. This randomized rounding process need not guarantee a feasible integral solution. If this is the case, the algorithm compares the cost of paying penalty p to that of a minimum node-weight path from s to t and chooses the cheaper among the two. That is, it either adds the nodes of such a path into the integral solution or pays p, whichever costs less.

Let f_u denote the fraction corresponding to u and f_p the fraction corresponding to penalty p. We denote by S the solution nodes added by the algorithm and

by P the set of pairs for which the algorithm pays their penalties. Both sets are initially empty. Let H be the subgraph induced by S and let w_u denote the weight of node u. The steps of the algorithm upon the arrival of a new pair are depicted in Algorithm 1 below.

Algorithm 1.

Input: $G = (V, E)$, S and P, $\{s, t\}$ with penalty p
Output: S and P, such that either there is a path from s to t in the subgraph H induced by S or $\{s, t\} \in P$
If there is no path from s to t in H,
(i) While (maximum flow between s and t after edge-weight update) < 1,
- Construct a minimum cut between s and t.
Let Q be the set of its minimum cut nodes.
- For each node $u \in Q$, set: $f_u = f_u \cdot (1 + 1/w_u) + \frac{1}{|Q| \cdot w_u}$
- Set $f_p = f_p \cdot (1 + 1/p) + \frac{1}{|Q| \cdot p}$

(ii) Add every $u \in V$ with $f_u > \mu$ to S and add $\{s, t\}$ to P if $f_p > \mu$
(iii) If $\{s, t\} \notin P$ and there is no path from s to t in H, construct a minimum node-weight path from s to t. Let c be its cost. Add $\{s, t\}$ to P if $p < c$. Else, add each node in the path to S.

Correctness. The correctness of the algorithm follows immediately from the last step, which assures that for each arriving pair $\{s, t\}$ with penalty p, either its penalty p is paid or the nodes of a cheapest path connecting s to t are added into the solution. In the next section, we show that this algorithm has an $\mathcal{O}(\log n \log k)$-competitive ratio for OPC-NWSF.

4 Competitive Analysis

In this section, we analyze the performance of Algorithm 1 upon its termination. We show that it has an $\mathcal{O}(\log n \log k)$-competitive ratio.

We will refer to a node u or pair with penalty p by a single parameter x and denote by c_x the corresponding cost w_u or p. Let S and P represent the solution constructed by the algorithm, where S denotes the set of nodes in the integral solution and P denotes the set of pairs whose penalties are paid by the algorithm. Similarly, we let S' and P' represent the output of an optimal offline solution.

Lemma 1. *The cost of the fractional solution constructed by the algorithm is at most $\mathcal{O}(\log n)$ times the cost of the optimal offline solution.*

Proof. The proof is based on the following:

(a) Every fractional increase adds at most 2 to the fractional solution.

Proof. Assume the algorithm enters the while loop to serve a given pair and

constructs a minimum cut. Let K be the set containing the corresponding minimum cut nodes and the penalty corresponding to the given pair. The fractional cost added by each $i \in K$ is $\left(c_i \cdot (\frac{f_i}{c_i} + \frac{1}{|K| \cdot c_i})\right)$. Note that before the algorithm performs a fractional increase, we have that $\sum_{i \in K} f_i \leq 1$. In total, the fractional cost is thus:

$$\sum_{i \in K} c_i \cdot \left(\frac{f_i}{c_i} + \frac{1}{|K| \cdot c_i}\right) \leq 2 \tag{1}$$

□

(b) Every time the algorithm makes an increase, there is at least one $j \in S' \cup P'$ whose fraction increases. After $\mathcal{O}(\log n) \cdot c_j$ increases, the fraction of j becomes at least 1.

Proof. In order to have served a given pair, an optimal offline algorithm must have either paid the corresponding penalty or must contain at least one of the minimum cut nodes. Hence, there must be at least one $j \in S' \cup P'$ in every fractional increase. Moreover, the fraction of j becomes at least 1 after $\mathcal{O}(\log |K|) \cdot c_j$ weight increases, where $|K| \leq n$. □

(c) The fraction of j is not increased again by the algorithm.

Proof. We have two possibilities here. (1) $j \in P'$: assume for contradiction that the algorithm decides to increase the fraction of j. Since this fraction is already at least 1, the while condition will not be satisfied (maximum flow is at least 1 because of the virtual edge corresponding to j which belongs to every cut) and the algorithm would have now served the demand. (2) $j \in S'$: assume the maximum flow is less than 1 and the algorithm enters the while loop. Then j cannot be a minimum cut node for any minimum cut computed inside the while loop since its fraction is at least 1 (Min-cut Max-flow Theorem). □

Lemma 2. *The cost of the integral solution constructed by the algorithm is at most* $\mathcal{O}(\log k)$ *times the cost of the fractional solution.*

Proof. To serve a given pair $\{s, t\}$, the algorithm adds nodes to S (and/or pairs to P) in steps ii and iii. The expected cost of the integral solution in step ii can be upper bounded by $\mathcal{O}(\log k)$. Fix a $q : 1 \leq q \leq 2\lceil \log(k+1) \rceil$ and let i represent any node (or penalty). Let $\mathcal{L}$ be the set of all nodes and penalties ($|\mathcal{L}| = n + k$). Let $X_{i,q}$ be the indicator variable of the event that i is chosen by the algorithm. The expected cost of the integral solution is then at most:

$$\sum_{i \in \mathcal{L}} \sum_{q=1}^{2\lceil \log(k+1) \rceil} c_i \cdot Exp\left[X_{i,q}\right] \leq 2\left\lceil \log(k+1) \right\rceil \cdot \sum_{i \in \mathcal{L}} c_i \cdot f_i \tag{2}$$

As for the expected cost of the integral solution in step iii, we show that it is negligible. The algorithm performs step iii only if the given pair $\{s, t\}$ is not

served in previous steps. Assume C is any minimum cut of G between s and t after exiting the while loop. Let K be the set containing the corresponding minimum cut nodes of C and pair $\{s,t\}$. Note that the algorithm assures that $\sum_{i\in K} f_i \geq 1$ upon exiting the while loop. Thus, we have that for a single $1 \leq q \leq 2\lceil\log(k+1)\rceil$, the probability that $\{s,t\}$ is not served is at most:

$$\prod_{i\in K}(1-f_i) \leq e^{-\sum_{i\in K} f_i} \leq 1/e$$

Therefore, for all $1 \leq q \leq 2\lceil\log(k+1)\rceil$, the probability that $\{s,t\}$ is not served is at most $1/k^2$. Moreover, the algorithm serves $\{s,t\}$ by paying the cheapest cost possible. This cost is a lower bound on the cost of the optimal offline solution *Opt*. The overall expected cost for all k terminal pairs is thus $k \cdot 1/k^2 \cdot Opt$. □

Lemmas 1 and 2 ultimately lead to the following theorem.

Theorem 1. *There is an online polynomial-time $\mathcal{O}(\log n \log k)$-competitive randomized algorithm for the Online Prize-collecting Node-weighted Steiner Forest problem (OPC-NWSF), which is optimal up to constant factor.*

Corollary 1. *There is an online polynomial-time $\mathcal{O}(\log n \log k)$-competitive randomized algorithm for the Online Prize-collecting Node-weighted Steiner Tree problem (OPC-NWST), which is optimal up to constant factor.*

Corollary 2. *There is an online polynomial-time $\mathcal{O}(\log n \log k)$-competitive randomized algorithm for the Online Node-weighted Steiner Forest problem (ONWSF), which is optimal up to constant factor.*

5 Open Problems

We have presented a randomized online algorithm for the *Online Prize-collecting Node-weighted Steiner Forest* problem (OPC-NWSF), which is optimal up to constant factor. The very next question would be to find out whether one can design an optimal *deterministic* algorithm for the problem. This seems to be challenging even for the simplest variant, the *Online Node-weighted Steiner Tree* problem. In fact, most online approaches for *node-weighted* Steiner problems are randomized.

OPC-NWSF in planar graphs admits an $\mathcal{O}(\log^2 n)$-competitive ratio due to Hajiaghayi *et al.* [8]. The question is whether one can achieve a better competitive ratio for this problem, by incorporating, for instance, ideas from our algorithm to that of Hajiaghayi *et al.*.

Steiner problems have been recently studied with a *leasing* aspect in which rather than buying edges and being able to use them forever, edges are leased for different durations and costs. Connectivity requirements need to be satisfied for a limited period of time and edges can be used only when leased. The leasing

variants of the *Online Steiner Tree* and *Online Steiner Forest* problems in edge-weighted graphs have been studied by Bienkowski *et al.* [6] and Meyerson [12], respectively. It would be interesting to consider the node-weighted as well as the prize-collecting counterparts of these problems.

References

1. Alon, N., Awerbuch, B., Azar, Y.: The online set cover problem. In: Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing, STOC 2003, pp. 100–105. ACM, New York (2003)
2. Alon, N., Awerbuch, B., Azar, Y., Buchbinder, N., (Seffi) Naor, J.: A general approach to online network optimization problems. ACM Trans. Algorithms **2**(4), 640–660 (2006)
3. Angelopoulos, S.: The node-weighted steiner problem in graphs of restricted node weights. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 208–219. Springer, Heidelberg (2006). https://doi.org/10.1007/11785293_21
4. Awerbuch, B., Azar, Y., Bartal, Y.: Online generalized Steiner problem. Theor. Comput. Sci. **324**(2–3), 313–324 (2004)
5. Berman, P., Coulston, C.: Online algorithms for Steiner tree problems (extended abstract). In: Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, 4–6 May 1997, pp. 344–353 (1997)
6. Bienkowski, M., Kraska, A., Schmidt, P.: A deterministic algorithm for online steiner tree leasing. Algorithms and Data Structures. LNCS, vol. 10389, pp. 169–180. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62127-2_15
7. Hajiaghayi, M.T., Liaghat, V., Panigrahi, D.: Online node-weighted Steiner forest and extensions via disk paintings. In: 54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, Berkeley, CA, USA, 26–29 October 2013, pp. 558–567 (2013)
8. Hajiaghayi, M.T., Liaghat, V., Panigrahi, D.: Near-optimal online algorithms for prize-collecting Steiner problems. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8572, pp. 576–587. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43948-7_48
9. Imase, M., Waxman, B.M.: Dynamic Steiner tree problem. SIAM J. Discrete Math. **4**(3), 369–384 (1991)
10. Klein, P., Ravi, R.: A nearly best-possible approximation algorithm for node-weighted Steiner trees. J. Algorithms **19**(1), 104–115 (1995)
11. Korman, S.: On the use of randomization in the online set cover problem. Master's thesis, Weizmann Institute of Science, Israel (2005)
12. Meyerson, A.: The parking permit problem. In: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), Pittsburgh, PA, USA, 23–25 October 2005, pp. 274–284 (2005)
13. (Seffi) Naor, J., Panigrahi, D., Singh, M.: Online node-weighted Steiner tree and related problems. In: Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Washington, DC, USA. IEEE Computer Society, pp. 210–219 (2011)

14. Qian, J., Williamson, D.P.: An $O(\log n)$-competitive algorithm for online constrained forest problems. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6755, pp. 37–48. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22006-7_4
15. Schroeder, J., Guedes, A., Duarte Jr., E.P.: Computing the minimum cut and maximum flow of undirected graphs. Technical report, Department of Informatics, Federal University of Paraná (2004)

Median of 3 Permutations, 3-Cycles and 3-Hitting Set Problem

Robin Milosz[1,2], Sylvie Hamel[1(✉)], and Adeline Pierrot[2]

[1] Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Québec, Canada
{robin.milosz,sylvie.hamel}@umontreal.ca
[2] Laboratoire de Recherche Informatique, Université Paris-Sud, Orsay, France
adeline.pierrot@lri.fr

Abstract. The median of permutations problem consists in finding a consensus permutation of a given set of m permutations of size n. This consensus represent the "closest" permutation to the given set under the Kendall-tau distance. Since the complexity of this problem is still unknown for sets of 3 permutations, in the following work, we investigate this specific case and show an interesting link with the 3-Hitting Set problem.

1 Introduction

The problem of aggregating multiple rankings into one consensus ranking was already looked at two centuries ago [8] but was mostly studied in the last twenty years under different names and in different research areas: *rank aggregation problem* [1,11,12], *Kemeny rank aggregation* [2,4,9], *Kemeny-Young method* [17], *median ranking problem* [7] and *preference aggregation* [10]. Applications include determining the winner in a sport competition, deriving voting preferences for an election or aggregating answers returned by several Web engines.

From a theoretical point-of-view, if the rankings are on strictly ordered elements, rankings are *permutations* and the problem becomes that of finding the *median* of a set of permutations under a given distance. For the Kendall-tau distance [16], the problem of finding medians of a set of m permutations has been widely studied either by deriving some exact solvers [2,9], some approximation algorithms [21], some fixed-parameters algorithms [13,15,20] or working on space reduction techniques [4,6,19].

The problem has been proved to be NP-complete for sets of $m \geq 4$ permutations, m even [11] (some corrections of the proof was done in [5]). It was recently found [3] that it is NP-Hard for sets of $m \geq 7$ permutations, m odd. The theoretical complexity of the cases $m = 3$ and $m = 5$ remains open, thus making it relevant to investigate those two cases.

Supported by NSERC through an Individual Discovery Grant (Hamel), by FRQNT through a Ph.D's scholarship and Mitacs through a Globalink Research Award (Milosz).

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 224–236, 2018.
https://doi.org/10.1007/978-3-319-94667-2_19

Here, we focus on the case where $m = 3$, i.e. we are interested to find medians of sets of three permutations. We demonstrate in this work, that in order to solve the median of three permutations problem, one can only consider 3-cycles present in the majority graph associated with the three permutations. We further make the link with the 3-Hitting Set problem (one of Karp's 21 NP-complete problems [14]) which consist in finding a minimal set of elements that cover a collection of subsets, in our case covering the collection of all 3-cycles present in the majority graph.

2 Basic Definitions

A *permutation* π is a bijection of $[n] = \{1, 2 \ldots, n\}$ onto itself. The set of all permutations of $[n]$ is denoted $\mathcal{S}_n$. As usual we denote a permutation π of $[n]$ as $\pi = \pi_1 \pi_2 \ldots \pi_n$. The *position* of an element i in a permutation π is π_i^{-1}, where π^{-1} is the usual inverse of π under composition. Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of permutations of $[n]$, we will denote its *cardinality* by $\#\mathcal{R}$. We define the *order* between two elements i and j, $1 \leq i, j \leq n$, in a permutation π of $[n]$ as i *to the left of* j (resp. *to the right of*), denoted $i \prec_\pi j$ (resp. $j \prec_\pi i$), if $\pi_i^{-1} < \pi_j^{-1}$ (resp. $\pi_i^{-1} > \pi_j^{-1}$). For $\mathcal{R} \subset \mathcal{S}_n$, the *majority order* for elements i and j, $1 \leq i, j \leq n$ is $i \prec_\mathcal{R} j$ (resp. $j \prec_\mathcal{R} i$) if $\#\{\pi \in \mathcal{R} \mid i \prec_\pi j\} \underset{(resp. <)}{>} \#\{\pi \in \mathcal{R} \mid j \prec_\pi i\}$. We say that $j \prec_\mathcal{R} i$ is the *minority order* if $i \prec_\mathcal{R} j$ is the majority order. Note that when $\#\mathcal{R}$ is even, there can exist elements i and j, $1 \leq i, j \leq n$, for which neither a majority or a minority order is defined. The *Kendall-tau distance*, denoted Kt, counts the number of order disagreements between pairs of elements of two permutations $\pi, \sigma \in \mathcal{S}_n$ and is defined as $Kt(\pi, \sigma) = \#\{(i, j) | i < j \text{ and } [(i \prec_\pi j \text{ and } j \prec_\sigma i) \text{ or } (j \prec_\pi i \text{ and } i \prec_\sigma j)]\}$. Given any set of permutations $\mathcal{R} \subseteq \mathcal{S}_n$ and a permutation $\pi \in \mathcal{S}_n$, the *Kemeny score* is defined as $K(\pi, \mathcal{R}) = \sum_{\sigma \in \mathcal{R}} Kt(\pi, \sigma)$.

The *median of permutations* problem is stated as follows: Given $\mathcal{R} \subseteq \mathcal{S}_n$, we want to find a permutation $\pi^* \in \mathcal{S}_n$ such that $K(\pi^*, \mathcal{R}) \leq K(\pi, \mathcal{R}),\ \forall\ \pi \in \mathcal{S}_n$. Such a permutation π^* is called a *median permutation*. Note that a set $\mathcal{R}$ can have more than one median.

Finally, we define the *left matrice* $L(\mathcal{R})$ such that for any two elements $1 \leq i, j \leq n$, $L_{ij}(\mathcal{R}) = \#\{\pi \in \mathcal{R} \mid i \prec_\pi j\}$. It represents the number of time $i \prec j$ in permutations of $\mathcal{R}$. Note that $L_{xy}(\mathcal{R}) = \#\mathcal{R} - L_{yx}(\mathcal{R})$ and that the Kemeny score can be computed using the left matrice as $K(\pi, \mathcal{R}) = \sum_{1 \leq i, j \leq n | i \prec_\pi j} L_{ji}(\mathcal{R})$. Note that for ease of notation, we will write L_{ij} instead of $L_{ij}(\mathcal{R})$, when the context is clear.

3 3-Cycle Theorem

In this section, we put everything together to state and prove our main theorem on the ordering of pairs of elements in medians of sets of three permutations.

3.1 Definitions and Properties

First, let us set basic definitions and state some properties that will help understand and prove this main theorem called *3-cycle Theorem*. To do that, we need to build the following graph.

Definition 1. *Let $\mathcal{R} \subset \mathcal{S}_n$. The* **majority graph** *$G_{\mathcal{R}} = (V, E)$ of $\mathcal{R}$ is the weighted directed graph, with the set of vertices $V = \{i \,|\, 1 \leq i \leq n\}$ and the set of directed edges $E = \{(i,j) \,|\, i \prec_{\mathcal{R}} j$ is the majority order of $i \neq j \in V\}$. The* **weight** *of an edge (i,j) is $w_{(i,j)} = |\, L_{ij} - L_{ji} \,|$. Note that the majority graph is a weighted tournament graph when $\#\mathcal{R}$ is odd.*

Observatio 1. *Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of 3 permutations, then for every pair of elements i, j, $1 \leq i \neq j \leq n$, exactly one of the following holds:*

- *(i,j) is an edge of $G_{\mathcal{R}}$ of weight 3 (case $L_{ij} = 3$),*
- *(i,j) is an edge of $G_{\mathcal{R}}$ of weight 1 (case $L_{ij} = 2$),*
- *(j,i) is an edge of $G_{\mathcal{R}}$ of weight 1 (case $L_{ij} = 1$),*
- *(j,i) is an edge of $G_{\mathcal{R}}$ of weight 3 (case $L_{ij} = 0$).*

Definition 2. *We call* **LowerBound$_1$** *the trivial lower bound on the Kemeny Score of a median permutation of $\mathcal{R} \subset \mathcal{S}_n$, discussed in details in [10], and computed as follow:*

$$LowerBound_1(K(\pi, \mathcal{R})) = \sum_{\substack{i \neq j \in \{1,\ldots,n\} \\ i<j}} min\{L_{ij}(\mathcal{R}), L_{ji}(\mathcal{R})\}.$$

Definition 3. *A* **3-cycle** *(i,j,k) in a majority graph $G = (V, E)$ is a directed cycle of length three containing the edges (i,j),(j,k) and (k,i). The* **set of involved edges** *of G, denoted I_E, is the set of all edges in E that are contained in at least one 3-cycle.*

Example 1. *Figure 1 shows the majority graph of the set $\mathcal{R} = \{[4,5,1,2,3], [1,5,3,4,2], [5,2,3,4,1]\}$. In this instance, there are two 3-cycles: $(2,3,4)$ and $(3,4,1)$ and $I_E = \{(1,3),(2,3),(3,4),(4,1),(4,2)\}$.*

Proposition 1. *Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of 3 permutations and let $G_{\mathcal{R}} = (V, E)$ be its majority graph. Then if $e \in E$ is an edge of a 3-cycle of $G_{\mathcal{R}}$, $w_e = 1$.*

Proof. By contradiction, suppose that there is an edge (i,j) of weight 3 in a 3-cycle (i,j,k) of $G_{\mathcal{R}}$. Then in every permutation of $\mathcal{R}$, i is before j. Moreover, by definition of the majority graph, $k \prec_{\mathcal{R}} i$ and $j \prec_{\mathcal{R}} k$ have to be majority orders. Because $k \prec_{\mathcal{R}} i$ is a majority order, k has to be before i in at least 2 of the 3 permutations of $\mathcal{R}$. But i is always before j thus, by transitivity, k has to be before j in at least 2 of the 3 permutations. This is a contradiction because $j \prec_{\mathcal{R}} k$ is a majority order. Therefore, a 3-cycle can only have edges of weight 1. ■

Finding a median for a set $\mathcal{R} \subset \mathcal{S}_n$ of 3 permutations is equivalent to transforming the majority graph $G_\mathcal{R}$ into a direct acyclic graph (DAG) using the minimum number of edge inversions and taking into account the weights of those inverted edges [9,10]. Indeed, a permutation can be represented as a DAG (the majority graph of the set containing only this permutation), and all edges that are reversed with respect to $G_\mathcal{R}$ correspond to pairs of elements that are in their minority order in that permutation. Note that if the number of reversed edges is minimal, the resulting topological ordering of the graph is a median permutation. From [9], we have that the Kemeny score of a permutations is linked with $LowerBound_1$ and the weight of reversed edges with respect to $G_\mathcal{R}$:

Proposition 2 [9]**.** *Let $\mathcal{R} \subset \mathcal{S}_n$, let $G_\mathcal{R} = (V, E)$ be the majority graph of $\mathcal{R}$ and let $D(G_\mathcal{R})$ be a directed acyclic graph obtained by inverting edges in $G_\mathcal{R}$. Let π be the permutation obtained from the topological ordering of the vertices in $D(G_\mathcal{R})$, then:*

$$K(\pi, \mathcal{R}) = LowerBound_1 + \sum_{\substack{(i,j)\in E \\ s.t.(j,i)\in D(G_\mathcal{R})}} w_{(i,j)}.$$

Interestingly, we observed by simulations on sets of 3 permutations, that all edges of the majority graph that needed to be inverted to obtain a median permutation were contained in a 3-cycle of the original majority graph. This gives us the following theorem.

Theorem 1 (3-cycle Theorem). *Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of 3 permutations. Let $G_\mathcal{R} = (V, E)$ be its majority graph. Let π^* be any median permutation of $\mathcal{R}$. If an edge (i, j) of $G_\mathcal{R}$ is not contained in any 3-cycle, then $i \prec_{\pi^*} j$.*

Proof (Part 1). First, note that by Proposition 1, all edges (i, j), with $w_{(i,j)} = 3$ are not contained in any 3-cycle of $G_\mathcal{R}$. For these edges, the following theorem from [6], gives us the result.

Theorem 2 (Always Theorem [6]**).** *Let $\mathcal{R}$ be a set of m permutations, m odd. Let π^* be any median permutation of $\mathcal{R}$. If $i \prec_\pi j$, $\forall \pi \in \mathcal{R}$, then $i \prec_{\pi^*} j$.*

For those edges (i, j) of $G_\mathcal{R}$, not contained in any 3-cycle, with $w_{(i,j)} = 1$, we have to work a bit more. The proof of this case is detailed in Sect. 3.2. ■

Theorem 1 can drastically reduce the search space for a median permutation, since it allows us to fix the order of all pairs of elements not contained in any 3-cycle of the majority graph. In Fig. 1, all 3-cycles are represented with dashed edges.

3.2 Proof of the 3-Cycle Theorem

In the following section we prove Theorem 1 for the case where (i, j) of $G_\mathcal{R}$ is not contained in any 3-cycle, with $w_{(i,j)} = 1$, and where $\mathcal{R} \subset \mathcal{S}_n$ is a set of 3 permutations.

To help us do so, first note that we have the following partition.

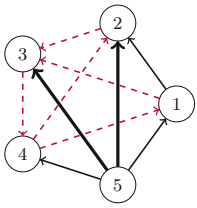


Fig. 1. The majority graph of the set $\mathcal{R} = \{[4,5,1,2,3],[1,5,3,4,2],[5,2,3,4,1]\}$. Bold edges have weight 3 and thin edges have weight 1. The 3-cycles are shown with dashed edges and the set of all those dashed edges is the set of involved edges I_E. With the 3-cycle theorem, the edge $(1,2)$ is preserved in any median permutation and should not be reversed despite being part of the cycle $(1,2,3,4)$. In this case, reversing edge $(3,4)$ results in a DAG with corresponding median permutation $\pi^* = [5,4,1,2,3]$.

Lemma 1. *Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of 3 permutations and let $G_\mathcal{R} = (V,E)$ be its majority graph. Let (i,j) be an edge of $G_\mathcal{R}$ not contained in any 3-cycle. Then the elements of $\mathcal{N}_{(i,j)} = \{1,2,\ldots,n\}\backslash\{i,j\}$ can be partitioned into the 5 distincts (disjoint) following subsets A, B, C, D and E:*

$$
\begin{aligned}
A &= \{x \in \mathcal{N}_{(i,j)} \,|\, L_{xi}(\mathcal{R}) = 3\},\\
B &= \{x \in \mathcal{N}_{(i,j)} \,|\, L_{xi}(\mathcal{R}) = 2 \text{ and } L_{xj}(\mathcal{R}) \geq 2\},\\
C &= \{x \in \mathcal{N}_{(i,j)} \,|\, L_{xi}(\mathcal{R}) \leq 1 \text{ and } L_{xj}(\mathcal{R}) \geq 2\},\\
D &= \{x \in \mathcal{N}_{(i,j)} \,|\, L_{xi}(\mathcal{R}) \leq 1 \text{ and } L_{xj}(\mathcal{R}) = 1\},\\
E &= \{x \in \mathcal{N}_{(i,j)} \,|\, L_{xj}(\mathcal{R}) = 0\}.
\end{aligned}
$$

Proof. This comes from the fact that for $x \in \mathcal{N}_{(i,j)}$, we cannot have $L_{xi}(\mathcal{R}) \geq 2$ ($x \prec_\mathcal{R} i$ is a majority order) and $L_{xj}(\mathcal{R}) \leq 1$ ($j \prec_\mathcal{R} x$ is a majority order), since this case correspond to having the 3-cycle (x,i,j) and we choose an edge (i,j) of $G_\mathcal{R}$ not contained in any 3-cycle. Since all of the other disjoint cases are represent by one of the subsets A, B, C, D and E, this conclude the proof. ■

Figure 2 gives two sets of three permutations, with elements in A, B, C, D and E.

$$
\mathcal{R}_1 = \left\{ \begin{array}{c} [\ b\ a\ \mathbf{i}\quad c\quad \mathbf{j}\ e\ d\ ] \\ [\ a\ \mathbf{i}\ d\quad c\quad b\ \mathbf{j}\ e\ ] \\ [\ \mathbf{j}\ e\ d\ c\ b\ a\ \mathbf{i}\ ] \end{array} \right\} \qquad \mathcal{R}_2 = \left\{ \begin{array}{c} [\ a\ b\ \mathbf{i}\quad c\quad \mathbf{j}\ d\ e\ ] \\ [\ a\ \mathbf{i}\ b\quad c\quad d\ \mathbf{j}\ e\ ] \\ [\ \mathbf{j}\ a\ b\ c\ d\ e\ \mathbf{i}\ ] \end{array} \right\}
$$

Fig. 2. Two examples of a set of three permutations having one element of each subset: $a \in A, b \in B, c \in C, d \in D$ and $e \in E$. The permutations are formatted to align elements i and j for better visualization.

To prove Theorem 1 by contradiction, we assume that there are two elements i and j such that (i, j) is an edge of the majority graph of $\mathcal{R}$ not contained in any 3-cycle, but there is a median permutation π^* in which $j \prec_{\pi^*} i$. We will concentrate on the set K of elements lying strictly between j and i in π^* (i.e. $\pi^* = ...jKi...$) and show that swapping j and i reduces the Kemeny score leading to a contradiction in our choice of median. Our proof is based on the following two lemmas, where Lemma 2 gives the possible values of $L_{xy}(\mathcal{R})$, for $x, y \in K \cup \{i, j\}$, and Lemma 3 shows that for $k \in K$, $k \notin A$ and $k \notin E$.

Lemma 2. *Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of 3 permutations and let $G_{\mathcal{R}} = (V, E)$ be its majority graph. Let (i, j) be an edge of weight $w_{(i,j)} = 1$ in $G_{\mathcal{R}}$ not contained in any 3-cycle. Let $j \prec_{\pi^*} i$, where $\pi^* \in \mathcal{S}_n$ is an assumed median of $\mathcal{R}$. Let $K \subset \{1, 2, \dots n\}$ be the set of elements lying strictly between j and i in π^*. Then, the values $L_{xy}(\mathcal{R})$ for $x, y \in K \cup \{i, j\}$ are the ones given in Table 1:*

Table 1. This table gives the possible values of $L_{xy}(\mathcal{R})$ for $x, y \in K \cup \{i, j\}$ (i.e. the number of permutations of $\mathcal{R}$ having x to the left y), where K is partitioned into the subsets A, B, C, D and E of Lemma 1. The bold (resp. italicized) numbers corresponds to values of $L_{xy}(\mathcal{R}_1)$ (resp. of $L_{xy}(\mathcal{R}_2)$) of the left example from Fig. 2.

L_{xy}	$y \in A$	$y \in B$	$y \in C$	$y \in D$	$y \in E$	$y = i$	$y = j$
$x \in A$	***0***,1,2,3	**1**,2,*3*	**2**,*3*	**2**,*3*	**2**,*3*	***3***	***2***
$x \in B$	*0*,1,**2**	***0***,1,2,3	**1**,2,*3*	**1**,2,*3*	**2**,*3*	***2***	***2***
$x \in C$	*0*,**1**	*0*,1,**2**	***0***,1,2,3	**1**,2,*3*	**2**,*3*	***1***	***2***
$x \in D$	*0*,**1**	*0*,1,**2**	*0*,1,**2**	***0***,1,2,3	**1**,2,*3*	***1***	***1***
$x \in E$	*0*,**1**	*0*,**1**	*0*,**1**	*0*,1,**2**	***0***,1,2,3	***1***	***0***
$x = i$	***0***	***1***	***2***	***2***	***2***	-	***2***
$x = j$	***1***	***1***	***1***	***2***	***3***	***1***	-

Proof. First, Theorem 2 and the fact that $\pi^* = \dots jKi \dots$ is a median of $\mathcal{R}$ imply that $L_{kj}(\mathcal{R}) \neq 3$ and $L_{ik}(\mathcal{R}) \neq 3, \forall k \in K$. Combining this observation and the definitions of subsets A, B, C, D, E, give us that $L_{ai} = 3$, $\forall a \in A$, $L_{bi} = L_{bj} = 2$, $\forall b \in B$, $L_{ic} = L_{cj} = 2$, $\forall c \in C$, $L_{id} = L_{jd} = 2$, $\forall d \in D$, and $L_{je} = 3$, $\forall e \in E$.

Second, since (i, j) is an edge of weight $w_{(i,j)} = 1$ in $G_{\mathcal{R}}$, we know that the majority order between i and j is $i \prec_{\mathcal{R}} j$ with $L_{ij}(\mathcal{R}) = 2$. Since $\forall a \in A$, a is always left of i, by transitivity, every element $a \in A$ has the majority order $a \prec_{\mathcal{R}} j$ and $L_{aj} = 2$, $\forall a \in A$. Transitivity also gives us $L_{ak} \geq 2$, $\forall a \in A$ and $\forall k \in C \cup D \cup E$. Symmetrically, $L_{ie} = 2$, $\forall e \in E$ and $L_{ke} \geq 2$, $\forall e \in E$ and $\forall k \in A \cup B \cup C$.

Third, since $\forall b \in B$, $L_{bi}(\mathcal{R}) = 2$, we know that in one of the three permutations of $\mathcal{R}$, b is to the right of i. Combining this with the fact that $\forall a \in A$, a is

always to the left of i ($L_{ai} = 3$) gives us that $L_{ba} \neq 3$, $\forall a \in A, b \in B$. Similarly, we have $L_{cb} \neq 3$, $\forall b \in B, c \in C$, $L_{dc} \neq 3$, $\forall c \in C, d \in D$, $L_{ed} \neq 3$, $\forall d \in D, e \in E$ and $L_{db} \neq 3$, $\forall b \in B, d \in D$.

Finally, if we consider x and y to be elements of the same subset A, B, C, D or E, then we know nothing about their relative order in permutations of $\mathcal{R}$ leading, in that case to $L_{xy} \in \{0, 1, 2, 3\}$. ■

Lemma 3. *Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of 3 permutations and let $G_{\mathcal{R}} = (V, E)$ be its majority graph. Let (i, j) be an edge of weight $w_{(i,j)} = 1$ in $G_{\mathcal{R}}$ not contained in any 3-cycle. Let $\pi^* \in \mathcal{S}_n$ be an assumed median of $\mathcal{R}$ in which $j \prec_{\pi^*} i$. Let $K \subset \{1, 2, \ldots n\}$ be the set of elements lying strictly between j and i in π^*. Then, $\forall k \in K$, $L_{ki}(\mathcal{R}) \neq 3$ and $L_{kj}(\mathcal{R}) \neq 0$.*

Proof. First, note that if we partition K into the subsets A, B, C, D and E of Lemma 1, then for $k \in K$, showing that $L_{ki}(\mathcal{R}) \neq 3$ and $L_{kj}(\mathcal{R}) \neq 0$, is the same as showing that $k \notin A$ and $k \notin E$ i.e. subsets A and E are empty. To prove that A and E are empty, we will proceed in two steps. First, we will assume that they are not empty and show that we cannot have an element of E to the left of an element of A in our assumed median. Second, we will use this fact to conclude that A and E are empty.

Step 1. We want to show that there is no element of E to the left of an element of A in π^*. By contradiction, assume π^* has at least one element of E at the left of one element of A. Assume that $e \in E$ (resp. $a \in A$) is the rightmost (resp. leftmost) such element, so that there is neither elements of E nor elements of A lying between e and a in π^*. Let S be the sequence of elements lying between e and a in π^* (i.e. $\pi^* = ...j...eSa...i...$). Then S contains only elements of B, C and D. Let $B_s = B \cap S$ and $n_B = \#B_s$, and similarly for C and D. We show that we can build from π^* a permutation π' that reduces the Kemeny score.

If $n_D \geq n_B$ we take π' such that the position of every element in π' is the same as in π^*, except that a is moved to the left of S and e: $\pi' = ...j...aeS...i....$

Let Δ be the difference of Kemeny score between π' and π^*: $\Delta = K(\pi', \mathcal{R}) - K(\pi^*, \mathcal{R})$. Recall that for any permutation π, the Kemeny score $K(\pi, \mathcal{R}) = \sum_{1 \leq i,j \leq n | \pi_i^{-1} < \pi_j^{-1}} L_{ji}(\mathcal{R})$. Then

$$
\begin{aligned}
\Delta &= -L_{ae} + L_{ea} + \sum_{b \in B_s} (-L_{ab} + L_{ba}) + \sum_{c \in C_s} (-L_{ac} + L_{ca}) + \sum_{d \in D_s} (-L_{ad} + L_{da}) \\
&\overset{(1)}{<} -2 + 1 + \sum_{b \in B_s} (-1 + 2) + \sum_{c \in C_s} (-2 + 1) + \sum_{d \in D_s} (-2 + 1) \\
&= -1 + \sum_{b \in B_s} (1) + \sum_{c \in C_s} (-1) + \sum_{d \in D_s} (-1) \\
&= -1 + n_B - n_C - n_D \\
&\overset{(2)}{<} 0
\end{aligned}
$$

Inequality (1) comes from taking the bounding values of L in Lemma 2. Inequality (2) comes from the fact that $n_D \geq n_B$.

For $n_B > n_D$, if we take π' such that the position of every element in π' is the same as in π^*, except that e is moved to the right of S and a: $\pi' = ...j...Sae...i...$, we can show again, using the same kind of computation as before, that $\Delta = K(\pi', \mathcal{R}) - K(\pi^*, \mathcal{R}) < 0$. In both cases, the Kemeny score is reduced, contradicting that π^* is a median permutation. Therefore, no element of E can precede an element of A in π^*.

Step 2. Now, we show that A and E are empty. Assume by contradiction that A is non empty. Let a be the leftmost element of A in π^*. Let S be the sequence of elements lying between j and a in π^* (i.e. $\pi^* = ...jSa...i...$). From Step 1, S contains only elements of B, C and D. Let $B_s = B \cap S$ and $n_B = \#B_s$, and similarly for C and D. Again we show that we can build from π^* a permutation π' that reduces the Kemeny score: If $n_B > n_D$ we take π' such that the position of every element in π' is the same as in π^*, except that j is moved to the right of S and a: $\pi' = ...Saj...i...$. Let Δ be the difference of Kemeny score between π' and π^* : $\Delta = K(\pi', \mathcal{R}) - K(\pi^*, \mathcal{R})$. Then

$$\begin{aligned}
\Delta &= -L_{aj} + L_{ja} + \sum_{b \in B_s} (-L_{bj} + L_{jb}) + \sum_{c \in C_s} (-L_{cj} + L_{jc}) + \sum_{d \in D_s} (-L_{dj} + L_{jd}) \\
&\overset{(1)}{=} -2 + 1 + \sum_{b \in B_s} (-2 + 1) + \sum_{c \in C_s} (-2 + 1) + \sum_{d \in D_s} (-1 + 2) \\
&= -1 + \sum_{b \in B_s} (-1) + \sum_{c \in C_s} (-1) + \sum_{d \in D_s} (1) \\
&= -1 - n_B - n_C + n_D \\
&\overset{(2)}{<} 0
\end{aligned}$$

Equality (1) comes from taking the values of L, given in Table 1. Inequality (2) comes from the fact that $n_B > n_D$.

For the case $n_D \geq n_B$, we move a at the immediate left of j (i.e. $\pi' = ...ajS...i...$) and we can show again, using the same kind of computation as before, that $\Delta = K(\pi', \mathcal{R}) - K(\pi^*, \mathcal{R}) < 0$. In both cases, the Kemeny score is reduced, contradicting that π^* is a median permutation. Therefore, A is empty. The proof that E is empty is symmetrical. ■

We now have everything we need to finish the proof of our Theorem 1 and conclude this section.

Proof of Theorem 1 (Part 2). Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of 3 permutations and let $G_{\mathcal{R}} = (V, E)$ be its majority graph. Let (i, j) be an edge of weight $w_{(i,j)} = 1$ in $G_{\mathcal{R}}$ not contained in any 3-cycle. Let $\pi^* \in \mathcal{S}_n$ be an assumed median of $\mathcal{R}$ with $j \prec_{\pi^*} i$. Let $K \subset \{1, 2, \ldots n\}$ be the set of elements lying strictly between j and i in π^* i.e. we have $\pi^* = ...jKi...$. From Lemma 3, K contains only elements of B, C and D. Let $\pi' = ...iKj...$ obtained from π^* by exchanging the positions of element i and element j. Let Δ be the difference of Kemeny score between π'

and π^*, i.e. $\Delta = K(\pi', \mathcal{R}) - K(\pi^*, \mathcal{R})$. Then

$$\begin{aligned}
\Delta &= -L_{ij} + L_{ji} + \sum_{b\in B}(-L_{ib} + L_{bi} - L_{bj} + L_{jb}) \\
&\quad + \sum_{c\in C}(-L_{ic} + L_{ci} - L_{cj} + L_{jc}) + \sum_{d\in D}(-L_{id} + L_{di} - L_{dj} + L_{jd}) \\
&\overset{(1)}{=} -2 + 1 + \sum_{b\in B}(-1 + 2 - 2 + 1) \\
&\quad + \sum_{c\in C}(-2 + 1 - 2 + 1) + \sum_{d\in D}(-2 + 1 - 1 + 2) \\
&= -1 + \sum_{b\in B}(0) + \sum_{c\in C}(-2) + \sum_{d\in D}(0) \\
&= -1 + -2 \times \#C \\
&< 0
\end{aligned}$$

Equality (1) comes from taking the values of L given in Lemma 2. The permutation π' has a lower Kemeny score than the assumed median permutation π^*, leading to a contradiction. ■

From preliminary tests we observed that Theorem 1 seems to be also valid on sets of m permutations, m odd. Therefore we state the following conjecture:

Conjecture 1. *Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of m permutations, m odd. Let $G_\mathcal{R} = (V, E)$ be its majority graph. Let π^* be any median permutation of $\mathcal{R}$. If an edge (i, j) of $G_\mathcal{R}$ is not contained in any 3-cycle, then $i \prec_{\pi^*} j$.*

The 3-Cycle Theorem has an important significance since we can prove that pairs of elements that are ordered by previous methods such as an extension of the Condorcet Criterion [8], the Major Order Theorem [19], the Always Theorem [6] and the 3/4 - Majority Rule [4], the two last ones being equivalent in the case of three permutations, are also ordered by Theorem 1. When combined with an ILP solver this space reduction greatly improves the solving time for randomly generated data as well as for real data (see supplementary material[1]).

4 Link with the 3-Hitting Set Problem

The Hitting Set problem (HSP) is defined as follows: Let $\mathcal{E}$ be our set of elements and T a set of subsets of $\mathcal{E}$. Find the minimum cardinality subset $S \subseteq \mathcal{E}$ such that every subset of T contains at least one element of S.

The **3-Hitting Set problem (3HS)** is the case where all subsets in T are of size 3. Both the Hitting Set problem and the 3-Hitting Set problem are NP-Hard [14]. The 3-Hitting Set problem can be formulated in Integer Linear Programming (ILP) as:

$$minimize : \sum_{e\in\mathcal{E}} x_e$$

Subject to:

$$x_{e_{t1}} + x_{e_{t2}} + x_{e_{t3}} \geq 1, \quad \forall\{t_1, t_2, t_3\} \in T,$$

[1] http://www.iro.umontreal.ca/~hamelsyl/M3P_3Cycles_3HS.html.

$$x_e \in \{0,1\}, \quad \forall e \in \mathcal{E}.$$

Where x_e is the binary variable that indicates if element e is included in the solution subset $S \subseteq \mathcal{E}$. The first constraint forces the solution to cover every subset $\{t_1, t_2, t_3\}$ of T with a least one element. The last constraint make the variables x_e binary.

Let S be the solution (the set of selected elements) of the 3-Hitting Set problem and $\#S$ its "value" as the score of the objective function, i.e. here its cardinality.

The link between the 3-Hitting Set problem and the median of a set $\mathcal{R} \subset \mathcal{S}_n$ of three permutations is made by observing that for every 3-cycle of $G_\mathcal{R} = (V, E)$, at least one edge has to be reversed to obtained a permutation. Therefore, the minimum number of edges covering all 3-cycles is a lower bound on the number of edges needed to be reverse to make $G_\mathcal{R}$ a DAG. In order to cover all 3-cycles using the 3HS, we choose $\mathcal{E} = I_E$, the set of involved edges of the majority graph $G_\mathcal{R} = (V, E)$ (see Definition 3), and T, the set of 3-cycles of $G_\mathcal{R}$ (see Fig. 3).

Proposition 3. *Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of 3 permutations, $LowerBound_1$ be the trivial lower bound of Definition 2 and $G_\mathcal{R} = (V, E)$ be the majority graph of $\mathcal{R}$. Let T be the set of all 3-cycles of $G_\mathcal{R}$ and I_E, the set of its involved edges. If S is a solution of value $\#S$ of the 3HS problem applied on (I_E, T) then:*

$$LowerBound_1 + \#S \leq K(\pi^*, \mathcal{R}) \leq K(\pi, \mathcal{R}) \ \forall \pi \in S_n,$$

where π^* is a median permutation of $\mathcal{R}$.

Proof. From Theorem 1, only edges contained in 3-cycles can be reversed to make $G_\mathcal{R}$ a DAG with a minimal number of reversed edges. Since $\#S$ is the minimal number of edges to select in order to cover all sets of T in the 3HS, we have that $G_\mathcal{R}$ cannot become a DAG with less than $\#S$ reversed edges.

From Observation 1 and Proposition 1, we have that the cost of reversing the edge (i, j) of a 3-cycle in $G_\mathcal{R}$ is 1.

From Proposition 2, we have that the Kemeny Score of a permutation is the sum of the $LowerBound_1$ and the number of reversed edges in $G_\mathcal{R}$ times their weight. In the case of a median of three permutations, the weight of edges in 3-cycles is always 1. Then, any reversed edge contributes 1 to the Kemeny Score. Since $\#S$ is a lower bound on the number of reversed edges, we have that $LowerBound_1 + \#S$ is a lower bound for the Kemeny score of a median permutation. ■

This result greatly improves, for this $\#\mathcal{R} = 3$ case, the two lower bounds based on linear programming and cycles given in [9], which had to consider all cycles present in the majority graph.

Through preliminary empirical observations, we noted that the lower bound $LowerBound_1 + \#S$ seems to be a really tight lower bound: in all our tests we have the surprising observation of $LowerBound_1 + \#S = K(\pi^*, \mathcal{R})$, with π^* a median permutation of $\mathcal{R} \subset \mathcal{S}_n$, so we conjecture the following:

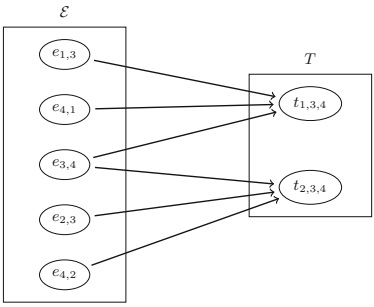


Fig. 3. 3-Hitting Set problem with $\mathcal{E}$ as the set of involved edges I_E and T the set of 3-cycles from the example in Fig. 1. In this case, selecting edge $e_{3,4}$ covers all 3-cycles thus $S = \{e_{3,4}\}$ and $\#S = 1$. Reversing this edge in the original majority graph makes the graph of Fig. 1 acyclic.

Conjecture 2. *Let $\mathcal{R} \subset \mathcal{S}_n$ be a set of 3 permutations, $LowerBound_1$ be the trivial lower bound of Definition 2 and $G_\mathcal{R} = (V, E)$ be the majority graph of $\mathcal{R}$. Let T be the set of all 3-cycles of $G_\mathcal{R}$ and I_E, the set of its involved edges. If S is a solution of value $\#S$ of the 3HS problem applied on (I_E, T) then:*

$$LowerBound_1 + \#S = K(\pi^*, \mathcal{R}) \leq K(\pi, \mathcal{R}) \ \forall \pi \in S_n,$$

where π^* is a median permutation of $\mathcal{R}$.

The conjecture was extensively tested on 10000 sets of 3 uniform random permutations of size $n \in \{10, 20, 30, 40, 50\}$ and 70 sets of permutations from PrefLib.org [18] (with $10 \leq n \leq 90$). No counter-example has been found. The conjecture can be extended for the cases where $\mathcal{R} \subset \mathcal{S}_n$ is a set of m permutations, m odd, with the weighted 3-Hitting Set problem.

5 Perspectives

The theoretical complexity of the median of permutations problem on sets of three permutations is still unknown making it an interesting problem to investigate. In this article, we presented our 3-Cycle Theorem, that allows us to drastically reduce the search space for a median of 3 permutations by fixing the order of all pairs of elements not contained in any 3-cycle of the majority graph of the given set of permutations. We then conjecture that our 3-Cycle Theorem is still true for sets of m permutations, m odd. It would be very interesting to be able to prove the stated conjecture since its reach englobes previous methods such as the Extended Condorcet Criterion [8], the Major Order Theorem [19], the Always Theorem [6] and the 3/4 - Majority Rule [4].

We also stated a new tight lower bound on the problem and a conjecture linking the 3HS problem with the median of three permutations problem. In future works, it would be interesting to prove this conjecture and to investigate

the complexity of the median of 3 permutations problem using this link to the 3-Hitting Set problem which is known to be NP-Hard.

Acknowledgements. Thanks to Sarah Cohen-Boulakia, Alain Denise and Pierre Andrieu from the bioinformatic team of Laboratoire de Recherche Informatique of Université Paris-Sud for useful advices and thoughts. Thanks to Mitacs which made this collaboration possible through a Mitacs Globalink grant.

References

1. Ailon, N.: Aggregation of partial rankings, p-ratings and top-m lists. Algorithmica **57**(2), 284–300 (2010)
2. Ali, A., Meilă, M.: Experiments with Kemeny ranking: what works when? Math. Soc. Sci. **64**, 28–40 (2012)
3. Bachmeier, G., Brandt, F., Geist, C., Harrenstein, P., Kardel, K., Peters, D., Seedig, H.G.: k-Majority Digraphs and the Hardness of Voting with a Constant Number of Voters arXiv: http://arxiv.org/abs/1704.06304v1 (2017)
4. Betzler, N., Bredereck, R., Niedermeier, R.: Theoretical and empirical evaluation of data reduction for exact Kemeny rank aggregation. Auton. Agent. Multi-Agent Syst. **28**, 721–748 (2014)
5. Biedl, T., Brandenburg, F.J., Deng, X.: Crossings and permutations. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 1–12. Springer, Heidelberg (2006). https://doi.org/10.1007/11618058_1
6. Blin, G., Crochemore, M., Hamel, S., Vialette, S.: Median of an odd number of permutations. Pure Math. Appl. **21**(2), 161–175 (2011)
7. Cohen-Boulakia, S., Denise, A., Hamel, S.: Using medians to generate consensus rankings for biological data. In: Bayard Cushing, J., French, J., Bowers, S. (eds.) SSDBM 2011. LNCS, vol. 6809, pp. 73–90. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22351-8_5
8. Condorcet, M.J.: Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix. Imprimerie Royale, 191 p. (1785)
9. Conitzer, V., Davenport, A., Kalagnanam, J.: Improved bounds for computing Kemeny rankings. In: Proceedings of AAAI - Volume 1, pp. 620–626 (2006)
10. Davenport, A., Kalagnanam, J.: A computational study of the Kemeny rule for preference aggregation. In: Proceedings of the 19th National Conference on Artificial Intelligence, AAAI 2004, pp. 697–702 (2004)
11. Dwork, C., Kumar, R., Naor, M., Sivakumar, D.: Rank aggregation methods for the web. In: proceedings of the 10th WWW, pp. 613–622 (2001)
12. Fagin, R., Kumar, R., Mahdian, M., Sivakumar, D., Vee, E.: Comparing partial rankings. SIAM J. Discret. Math. **20**(3), 628–648 (2006)
13. Guo, J., Niedermeier, R., Betzler, N., Fellows, M.R., Rosamond, F.A.: How similarity helps to efficiently compute Kemeny rankings. In: Proceedings of the 8th International Conference on Autonomous Agents and Multi-Agent Systems (2009)
14. 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/10.1007/978-1-4684-2001-29

15. Karpinski, M., Schudy, W.: Faster algorithms for feedback arc set tournament, Kemeny rank aggregation and betweenness tournament. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010. LNCS, vol. 6506, pp. 3–14. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17517-6_3
16. Kendall, M.: A new measure of rank correlation. Biometrika **30**, 81–89 (1938)
17. Young, H.P., Levenglick, A.: A consistent extension of Condorcet's election principle. SIAM J. Appl. Math. **35**(2), 285–300 (1978)
18. Mattei, N., Walsh, T.: PrefLib: a library for preferences http://www.preflib.org. In: Perny, P., Pirlot, M., Tsoukiàs, A. (eds.) ADT 2013. LNCS (LNAI), vol. 8176, pp. 259–270. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41575-3_20
19. Milosz, R., Hamel, S.: Medians of permutations: building constraints. In: Govindarajan, S., Maheshwari, A. (eds.) CALDAM 2016. LNCS, vol. 9602, pp. 264–276. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29221-2_23
20. Nishimura, N., Simjour, N.: Parameterized enumeration of (locally-) optimal aggregations. In: Dehne, F., Solis-Oba, R., Sack, J.-R. (eds.) WADS 2013. LNCS, vol. 8037, pp. 512–523. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40104-6_44
21. Schalekamp, F., van Zuylen, A.: Rank aggregation: together we're strong. In: Proceedings of the 11th SIAM Workshop on Algorithm Engineering and Experiments, ALENEX, pp. 38–51 (2009)

On the Parameterized Complexity of Colorful Components and Related Problems

Neeldhara Misra(✉)

Indian Institute of Technology, Gandhinagar, India
neeldhara.m@iitgn.ac.in
http://www.neeldhara.com

Abstract. The colorful components framework is motivated by applications emerging from computational biology. A vertex-colored graph G is said to be colorful if every color appears exactly once. The general goal is to remove a collection of edges from an undirected vertex-colored graph G such that in the resulting graph H all the connected components are colorful. We want H to optimize an appropriate objective function. Two natural functions involve deleting the smallest number of edges (which we refer to as Colorful Components) and maximizing the number of edges in the transitive closure of the remaining components (which we refer to as MEC).

These problems are well-studied from the point of view of classical complexity, approximation algorithms, and parameterized algorithms. We complement and improve on some of the results in the literature concerning MEC and Colorful Components. In the context of MEC, we demonstrate a linear kernel on trees and a randomized $k^{O(k)}$ algorithm, where k is the standard parameter. Both of these results directly improve on previously known results about the problem. For the Colorful Components problem, we demonstrate a FPT algorithm for the vertex cover parameter, which is a well-motivated structural parameterization given that the problem is already para-NP-hard when parameterized by a deletion set to a disjoint union of stars.

1 Introduction

We consider some problems that arise in the framework of *colorful components*, where we are given a vertex-colored graph G and the goal is to find a subgraph H that induces colorful components — in other words, no color appears more than once in any connected component of H. Usually, one is interested in the "largest" such subgraph possible, and this notion manifests in a few different optimization objectives. For instance, we might ask for the smallest subset of edges that needs to be removed from G such that all connected components of

N. Misra The author acknowledges support by the INSPIRE Faculty Scheme, DST India (project IFA12-ENG-31).

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 237–249, 2018.
https://doi.org/10.1007/978-3-319-94667-2_20

$G \setminus F$ are colorful [8,9]. The dual objective would be to find the largest subset of edges F for which every connected component in $G[F]$ is colorful — this is commonly formulated as the problem of maximizing the number of edges in the transitive closure of a solution [16]. Other well-studied goals involve minimizing the total number components [2] or the number of singleton components in the subgraph [16].

Readers may recognize the version of the problem that asks for the smallest edge deletion set as a particular type of *cut problem*. Indeed, recall that in the MultiCut problem [5,15], we are given a graph and "demands" specified as pairs of vertices $\{(s_i, t_i)\}_{i=0}^{p}$, and the goal is to remove the smallest number of edges such that in the remaining graph, there is no path between s_i and t_i for any $i \in [p]$. Clearly, the first objective that we discussed is a special case of MultiCut: we can declare a demand corresponding to every pair of vertices that have the same color. Similarly, in the Multi-Multiway Cut problem [4], we are given a graph and a family of terminal sets specified by vertex subsets $S_1, S_2, \ldots, S_p$, and the goal is to remove the smallest number of edges that intersects every path between all pairs of vertices in S_i, for any $i \in [p]$. Again, it is easy to see that deleting the smallest subset of edges so that all remaining components are colorful is a special case of Multi-Multiway Cut: one where the terminal sets correspond to a partition of the entire vertex set.

The problems involving colorful components also carry the flavor of the Graph Motif problem [13], where the input is a vertex-colored graph and a multiset of colors M, usually called a *motif*, and the goal is to determine if there exists a connected subgraph H such that the set of colors used by the vertices of H corresponds exactly to M. When M is a set (rather than a multiset), M is called colorful. The problem of discovering motifs and colorful motifs in graphs are well-studied problems from both the theoretical and practical standpoints. Several variations of this theme have been explored in the literature, one of which is to ask for a colorful subgraph on at least k vertices, without fixing a motif. Both colorful components and graph motif family of problems are motivated by applications in computational biology. For instance, the former has been used as a model for problems such as *Multiple Sequence Alignment* [9] and *Network Alignment* for multiple protein-protein interaction (PPI) networks. We refer the reader to the discussions in [1,8,16] for a more detailed discussion of the computational biology background that has motivated the algorithmic studies of these problems.

Background and Our Contributions. We now summarize the context for the work in this contribution. We focus on the colorful components family of problems. First, we consider the problem of maximizing the number of edges in the colorful subgraph, which we refer to as the problem of Maximizing Edges in the Transitive Closure (MEC). This problem is known to be NP-hard. Recently, [12] studied the parameterized complexity of the problem when parameterized by the number of edges in the solution. They demonstrated that the

problem is FPT by showing an exponential kernel of size $O(k^2\sqrt{k}2^{2k})$ on general graphs. Further, they also obtained a quadratic kernel on trees. We complement these results by showing a simple randomized FPT algorithm using color coding that has a better asymptotic complexity, namely $O(k^k)$ when compared to a brute-force approach on the exponential kernel. Our approach here is to reduce this problem to one of finding enough colorful motifs that when combined correspond to the desired MEC solution.

We then turn our attention to the Colorful Components problem. In [8], the authors show that the problem is in P when the instance has at most two colors and is NP-hard otherwise, even on trees of diameter four, and separately, even when the maximum degree of the graph is bounded by a constant. On the other hand, they also show an algorithm with running time $2^c n^{O(1)}$ on trees with n vertices by dynamic programming and also an algorithm with running time $(c-1)^k m$ on general graphs with m edges. The authors also take advantage of a weighted multi-multi cut formulation leading to interesting consequences on the experimental front. The hardness of trees on diameter four shows that the problem is para-NP-hard when parameterized by the size of a deletion set to graphs that are a disjoint union of stars. Similarly, the hardness of the Minimum Colorful Components (where the goal is to minimize the number of connected components) shown by [12] can be adapted to show para-NP-hardness when parameterized by the size of a deletion set to graphs that are a disjoint union of paths. A natural goal, from the perspective of structural parameterizations, is to settle the complexity of Colorful Components when parameterized by the vertex cover of the graph G. We answer this question in the affirmative showing that the problem is FPT when parameterized by vertex cover.

Related Work. Apart from Colorful Components and Maximizing Edges in the Transitive Closure (MEC), other optimization objectives were studied by [1,2]. Experimental advances based on ILP methods for Colorful Components were performed in [7]. It is worth noting that although most variants of problems in this framework are NP-hard, the version that seeks to minimize the number of singleton components was shown to admit a polynomial-time algorithm [2]. Also, for a comprehensive study of MEC from the perspective of approximation algorithms and hardness, we refer to [1].

2 Problem Definitions and Notation

We use standard graph-theoretic notation following [11] unless mentioned otherwise. We refer the reader to [10] for an introduction to terminology and techniques concerning parameterized algorithms. We only mention here, for the sake of completeness, the key definitions relevant to this work. A parameterized problem is denoted by a pair $(Q, k) \subseteq \Sigma^* \times \mathbb{N}$. The first component Q is a classical language, and the number k is called the parameter. Such a problem is

fixed–parameter tractable (FPT) if there exists an algorithm that decides it in time $O(f(k)n^{O(1)})$ on instances of size n. A *kernelization algorithm* for a problem P is a polynomial-time algorithm that, given an instance Q of P and a parameter k, returns an instance Q' of P whose size is bounded by some function $f(k)$, such that Q is a YES-instance iff Q' is a YES-instance. We then say that P has a kernel of size $f(k)$. One of the FPT algorithms presented in this work follows the technique of color coding, a widely-employed tool introduced in [3].

We now summarize the problem definitions that we will need subsequently. Recall that a subgraph H of a graph G is *colorful* if no color appears more than once in any connected component of H. We now define the following problems concerning the discovery of colorful subgraphs that have different optimization objectives. The first problem is MAXIMIZING EDGES IN THE TRANSITIVE CLOSURE (MEC).

MAXIMIZING EDGES IN THE TRANSITIVE CLOSURE (MEC)
Input: A vertex-colored graph $(G = (V, E); c : V \to [\ell])$ and $k \in \mathbb{N}$.
Question: Does there exist a colorful subgraph H of G that has at least k edges in the transitive closure of its connected components?

The next problem that we consider is COLORFUL COMPONENTS.

COLORFUL COMPONENTS (CC)
Input: A vertex-colored graph $(G = (V, E); c : V \to [\ell])$ and $k \in \mathbb{N}$.
Question: Does there exist a subset $F \subseteq E$ of at most k edges such that every connected component in $G \setminus F$ is colorful?

3 A Simple FPT Algorithm for MEC

Let $(G = (V, E); c : V \to [\ell], k)$ be an instance of MEC. In this section, we focus on the question of whether G admits a solution with *exactly* k edges. Note that the original question can be addressed by simply repeating the algorithm described in this section for a suitable range of values of k: indeed, if G admits a solution F with more than k edges in the transitive closure of the connected components of $G[F]$, then G also admits a solution F' that has exactly k' edges for some $k \in [k, 5k]$. To see this, let F be a solution with strictly more than k edges in the transitive closure of its connected components. Modify F by removing edges one by one, and stop when the number of edges in the transitive closure falls below k for the first time. Let F' be the solution after adding back the last edge removed: we denote this edge by e. Let C be the component that e belonged to. Note that e must be a cut edge in the component C, else the number of edges in the transitive closure stays the same after removing e, contradicting our choice of e. Note that the connected components of $C \setminus \{e\}$, say X and Y, can have at most $\sqrt{2k}$ vertices each (and we remark here that this is a loose bound). Therefore,

the number of edges in the transitive closure of F' is at most $k + 4k = 5k$, where the first k edges is an upper bound on the number of edges in components of F' different from C and within X and Y while $4k$ is the bound for the number of new edges added to the transitive closure by the addition of the edge e.

We now turn to a description of our approach. To begin with, suppose (G, c, k) were a YES-instance of MEC, and further, let $F \subseteq E$ be an arbitrary but fixed solution. Let $C_1, \ldots, C_r$ be the vertex subsets corresponding to non-trivial connected components of $G[F]$. We refer to r as the *order* of F and define the *signature* of F as a tuple $\eth_F := (k_1, \ldots, k_r)$, where $k_i := \binom{|C_i|}{2}$. Note that $\sum_{i=1}^{r} k_i = k$.

We will also need to define the following graph motif problem, considered by [14]:

> COLORFUL GRAPH MOTIF (CGM)
> **Input:** A graph $G = (V, E)$, a set C, a function $\chi : V \to C$, an integer k.
> **Question:** Does there exist a subtree $T = (V_T, E_T)$ of G such that $|V_T| = k$ and for each $u, v \in V_T$ distinct, $\chi(u) \neq \chi(v)$?

The authors in [14] demonstrate a randomized algorithm running in time $O^\star(2^k)$ based on multilinear term detection. This problem can also be solved (albiet with a higher time complexity) by employing the color coding technique. For instance, see [6] for a description of this approach the special case when $k = |C|$. Our approach to an FPT algorithm is based on the color coding technique as well [3].

We guess the order of a possible solution and also its signature, $\eth_F$, given by $(k_1, \ldots, k_r)$. Now consider a partition of $V(G)$ into r parts. We say that a partition $\mathcal{P} = (P_1, \ldots, P_r)$ of V is *good* for F if $C_i \subseteq P_i$. Notice that if we are given a good partition, then we can discover a solution F whose signature is given by $\eth_F$, by treating each part of the partition $\mathcal{P}$ as an instance of the COLORFUL GRAPH MOTIF problem. After guessing the order and the signature, we use a random partition and, in the analysis, we exploit the fact that a random partition is also good with reasonable probability. Finally, a solution is found by invoking the algorithm for CGM on each part with an appropriate target. We now turn to a more detailed description and analysis of this algorithm.

Good Partitions and Colorful Motifs. Assume that $\mathcal{P} = (P_1, \ldots, P_r)$ is a partition of V that is good for F. Let G_i denote $G[P_i]$. We claim that G admits a solution F with signature $\eth_F$, if and only if for all $1 \leqslant i \leqslant r$, the graph G_i contains a colorful motif on at least q_i vertices, where q_i is the integer for which $\binom{q_i}{2} = k_i$. This follows from the definition of a good partition and the fact colorful connected components whose transitive closure have k_i edges are also colorful motifs on at least q_i vertices, and vice versa. Note that $q_i = O(\sqrt{k_i})$.

Goodness of Random Partitions. For any solution F, we have that:

$$\sqrt{2k} + 1 \leqslant \sum_{i=1}^{r} |C_i| \leqslant 2k.$$

Observe that the extreme cases in the bound above correspond to scenarios where either F has only one connected component with k edges in its transitive closure, which implies that the component comprises of $O(\sqrt{k})$ vertices, or when F has k connected components, each of which induces an edge. With q_i's defined as before, note that $|C_i| = q_i$. We let $q := \sum_{i=1}^{r} q_i$. Consider now partitions of $V(G)$ into q parts. Noting that there are q^q distinct labelings of vertices in $\cup_{i=1}^{r} C_i$, only one of which corresponds to a good partition for F. Specifically, the number of partitions that are good for F are q^{n-q}, while the number of partitions that are not good for F are $(q^q - 1) \cdot q^{n-q}$. Let $\mathscr{P}$ be a random partition of $V(G)$, by which we mean that every vertex is assigned to each of the possible q parts with equal probability. Then, by the discussion in this paragraph, it follows that:

$$P(\mathscr{P} \text{ is a good partition for } F) \geqslant \frac{1}{q^q}$$

We can now repeat the process of generating a random partition $O(q^q)$ times to reduce the probability of error (namely that there exists a solution with signature $\mathfrak{s}_F$ and yet none of the partitions considered were good for F) to a constant:

$$P(\text{None of the } q^q \text{ partitions are good for } F) < \left(1 - \frac{1}{q^q}\right)^{q^q} \leqslant \frac{1}{e}.$$

Running Time. The number of signatures is loosely bounded by $O(k^k)$ and since $q \leqslant k$, the number of trials that we run for any fixed choice of signature is also bounded by $O(k^k)$. For each trial, we run the CGM algorithm at most k times, which incurs a running time $O^\star(2^k)$. Therefore, the overall running time is bounded by $k^{O(k)} n^{O(1)}$. We summarize this discussion with the following statement.

Theorem 1. MEC *admits a randomized* FPT *algorithm with running time* $k^{O(k)} n^{O(1)}$*, where* k *is the standard parameter.*

4 An Improved Kernel for MEC on Trees

We now turn our attention to the MEC problem restricted to trees. In [12], a quadratic kernel is established by the standard parameter. Here, we demonstrate that this can be improved to a linear kernel. Our overall approach uses ideas quite similar to the ones used for the quadratic kernel. Before stating the reduction

rules formally, we give a high-level summary of our algorithm. We begin by observing that if two or more sibling leaves have the same color, then it is enough to retain only one of them, since at most one of them can contribute to a non-trivial component in any solution. Thus, when we focus on the tree induced by the leaves and their parents, we see a disjoint union of colorful stars, and thus inferring that the number of leaves is bounded by k: otherwise we already have a solution. This also implies a bound on the number of branching vertices. We are now left with vertices of degree two. Our final observation is that if we focus on *all* the degree two vertices and if they collectively contribute more than $2k$ edges, then again we can extract a matching on k edges and we are have an easy YES-instance. Otherwise, the number of internal non-branching vertices is also bounded by $2k$, giving us the overall linear bound. We have to argue about the degree two vertices with some care since even if the number of edges is bounded, there may be isolated vertices: but this can be handled by "padding" the degree two paths with an appropriate branch or leaf neighbor. We now turn to a detailed description of these arguments.

Let (T, c, k) be an instance of MEC where T is a tree. Recall that we may assume, without loss of generality, that c is a proper coloring. Also, we may assume that T has at least one vertex of degree at least three, since T would otherwise be a path, and we could obtain a trivial kernel by solving MEC in polynomial time. For convenience, we choose an arbitrary vertex of degree at least three, say r, and assume for the rest of this discussion that T is rooted at r. We begin by employing the following reduction rule.

Reduction Rule 1. *If* u *and* v *are two leaves in* T *with a common parent* p *and* $c(u) = c(v)$*, then delete* u*.*

The safety of the reduction rule follows from the fact that any valid solution of MEC cannot contain both (p, u) and (p, v). Therefore, at least one of u or v in $T[F]$ belongs to a trivial connected component, and therefore does not contribute to the solution. Absuing notation, we use T to continue to denote the instance after it is reduced with respect to the reduction rule above. Let p be a vertex that is a parent of a leaf in T. Using the fact that c is a proper coloring and that T is reduced, observe that the subtree induced by $N[p]$ is colorful. We now make following claim, noting that for the rest of this discussion, we use T to refer to a reduced instance.

Proposition 1. *If the number of leaves of* T *is at least* k*, then* (T, c, k) *is a* YES*-instance of* MEC*.*

Proof. Let Q denote the set of leaves of the tree T and let $P \subseteq V$ consist of the set of vertices that are parents of a leaf vertex in T. Consider:

$$F := \{(u, v) \mid u \in P, v \in Q \text{ and } (u, v) \in E(T)\}.$$

Recall that $G[F]$ is a disjoint union of stars where each component is colorful. Since every leaf contributes at least one edge to F, the number of edges in the transitive closure of F is also at least k. □

The safety of the following reduction rule follows from Proposition 1.

Reduction Rule 2. *If* T *has at least* k *leaves, then return a trivial* Yes*-instance.*

Let T be an instance reduced with respect to the first two reduction rules, which implies, in particular, that T has at most k leaves. We now proceed by *marking* all the vertices in T that have degree at least three. Denoting the number of leaves by ℓ and the subset of marked vertices by M, we note that $|M| \leqslant \ell \leqslant k$, using the fact that the number of internal vertices that have degree at least three (sometimes referred to as "branching" vertices) is bounded above by the number of leaves in T.

Now, recalling that we use Q to denote the set of leaves of T, consider the graph $H_0 := T \setminus (M \cup Q)$. Note that every vertex in H_0 has degree at most two, and therefore H is a disjoint union of paths with some paths possibly having length zero. For any maximal path $P \in H_0$, let $\{v_1, \ldots, v_t\}$ be the vertices on P in their order of appearance on the path, arranged in increasing order of distance from r. Note that this ordering is well-defined since $r \in M$, and therefore the root vertex r does not appear on any path $P \in H_0$. Observe that v_t has at least one neighbor in $(M \cup Q)$ — indeed, if not, then v_t would be a leaf vertex of T, contradicting the situation that $v_t \in H_0$. Let $t(P)$ denote an arbitrarily chosen neighbor of v_t from $(M \cup Q)$: we refer to this vertex as the *tail* of the path P. We now make the following claim about tail vertices in T:

Proposition 2. *Let* $P_1 = \{v_1, \ldots, v_a\}$ *and* $P_2 = \{u_1, \ldots, u_b\}$ *be two distinct maximal paths in* H_0*. Then* $t(P_1) \neq t(P_2)$*.*

Proof. Assume, for the sake of contradiction, that $t(P_1) = t(P_2) = x$. Let y denote the least common ancestor of v_1 and u_1. Observe that the paths $yTv_1, v_1P_1v_a, x, u_bP_2u_1, u_1Ty$ constitute a cycle, contradicting the tree structure of T. □

Let H_1 denote the set of all vertices identified as tail vertices and define $H := H_0 \cup H_1$. We now claim that if H has enough edges then we can extract a valid MEC solution from it. This also leads us to our final reduction rule.

Proposition 3. *If* H *has at least* $2k$ *edges, then* (T, c, k) *is a* Yes*-instance of* MEC*.*

Proof. Observe that every connected component of H is a path of length at least one. Let $t_1, \ldots, t_r$ denote the lengths of the paths in P, noting that $\sum_{i=1}^{r} t_i \geqslant 2k$

by assumption. Construct F by choosing alternating edges from every path in H, we get a matching on at least $\sum_{i=1}^{r} \lceil \frac{t_i}{2} \rceil \geqslant k$ edges, as desired. □

Reduction Rule 3. *If* H*, constructed according to the description above, has at least* $2k$ *edges, then return a trivial* YES*-instance.*

We summarize our discussion here by arguing that any instance reduced with respect to all the reduction rules above is a linear kernel.

Theorem 2. MEC *admits a kernel of size* $O(k)$*.*

Proof. If the algorithm stops by returning a trivial instance then there is nothing to prove. Any other instance that is reduced with respect to Reduction Rules 1–3 has at most k marked vertices, at most k leaves, and at most $2k$ vertices that are neither marked nor leaves. Therefore, the total number of vertices is bounded by $4k$, implying the claim. □

5 Structural Parameterization: Vertex Cover

In this section, we focus on the COLORFUL COMPONENTS problem, parameterized by the size of a vertex cover of the input graph G. We denote this problem by CC$[\![\tau]\!]$:

> COLORFUL COMPONENTS/VERTEX COVER (CC$[\![\tau]\!]$)
> **Input:** A vertex-colored graph $(G = (V, E); c : V \to [\ell])$, a vertex cover $T \subseteq V$ of size t and $k \in \mathbb{N}$.
> **Parameter:** τ
> **Question:** Does there exist a subset $F \subseteq E$ of at most k edges such that every connected component in $G \setminus F$ is colorful?

As is standard for such parameterizations, we assume that a vertex cover T is given as a part of the input and we use t to denote the size of the given vertex cover. The goal of this discussion is to demonstrate that CC$[\![\tau]\!]$ is in FPT. Without loss of generality, we assume that G is connected.

We now turn to a description of the main steps in our FPT algorithm. Fix an instance (G, T, c, k) of CC$[\![\tau]\!]$. We denote by F an optimal solution, which we will refer to typically for the sake of analysis. To begin with, observe that in $G \setminus F$, any vertex $v \in V \setminus T$ that is not isolated belongs to a connected component that contains at least one vertex of T. In particular, the number of non-singleton connected components in $G \setminus F$ does not exceed t.

Our algorithm begins by guessing the intersection S of F with the edges in $G[T]$. For a fixed choice of S, let $\mathscr{C}_S := \{C_1^S, \ldots, C_r^S\}$ denote the connected components of $(G \setminus S)[T]$. When S is clear from the context, we drop the superscripts when

referring to the components. We also guess the number of non-singleton components induced by $G \setminus F$ and denote this by $\hbar$. We say that two components C_i and C_j are *equivalent* if they belong to the same connected component of $G \setminus F$. Note that this is, in fact, an equivalence relation that induces exactly $\hbar$ equivalence classes on $\mathscr{C}_S$. We guess a partition of $\mathscr{C}_S$ into $\hbar$ equivalence classes: in other words, we try all possible labelings χ of the components $C_1, \ldots, C_r$ with label set $[\hbar]$. The semantics here is that two components that have the same label belong to the same connected component of $G \setminus F$. For any $1 \leqslant i \leqslant \hbar$, we refer to the union of all connected components that have label i as the i^{th} *bag* of the vertex cover with respect to the guess (S, χ) and denote this set by $\mathscr{B}_i$.

At this point, we pause to reflect on the structure of a YES-instance of a MEC. As we noted before, in $G \setminus F$, every vertex in the independent set $V \setminus T$ either "joins" one of the bags that is fixed by our guess from above, or breaks away into a singleton component. For a vertex v in $V \setminus T$, let $\overline{d_i(v)}$ denote the number of edges incident on v whose other endpoint lies *outside* $\mathscr{B}_i$. Then, define the *cost* of v as the following $(\hbar + 1)$-tuple: $c(v) = \langle \overline{d_1(v)}, \ldots, \overline{d_i(v)}, \ldots, \overline{d_\hbar(v)}, d(v) \rangle$. Intuitively, the i^{th} coordinate of $c(v)$ quantifies the number of edges that need to be deleted for v to belong to the same component as the vertices of $\mathscr{B}_i$ in $G \setminus F$, while the last coordinate indicates the cost of making v isolated in $G \setminus F$.

As a warm-up, we address the special case when the number of colors ℓ is bounded. Group the vertices in $V \setminus T$ according to their neighborhoods in T. In particular, two vertices $u, v \in V \setminus T$ belong to the same group if and only if $N(u) = N(v)$. Notice that there are 2^t groups and ℓ colors. For $X \subseteq T$ and for $q \in [\ell]$, define $\mathscr{I}(X, q) \subseteq V \setminus T$ as the following set:

$$\mathscr{I}(X, q) = \{v \mid v \in V \setminus T, N(v) = X, c(v) = q\}.$$

Note that the number of sets $\mathscr{I}(X, q)$ is bounded by $2^t \ell$. Observe also that all vertices in $\mathscr{I}(X, q)$ have the same cost vector. Now, consider a choice of $X \subseteq T$ and for $q \in [\ell]$ for which $n_{X,q} := |\mathscr{I}(X, q)| > \hbar$. Let U be a subset of any $n_{X,q} - \hbar$ many vertices chosen from $\mathscr{I}(X, q)$. We claim that:

$$(G, T, c, k) \equiv (G \setminus U, T, c', k - d n_{X,q}),$$

where c' is the projection of c on the vertices of $V \setminus U$ and d is the degree of any vertex in $\mathscr{I}(X, q)$. Indeed, this follows from the fact that in any solution F that respects the guess (S, χ), at least $n_{X,q} - \hbar$ vertices must be isolated in $G \setminus F$: if not, by a pigeon-hole argument, we would have that two vertices of $\mathscr{I}(X, q)$ join the same bag, which would contradict the fact that the vertices in any bag belong to a colorful connected component. This leads to a contradiction, since any two vertices of $\mathscr{I}(X, q)$ have the same color.

So far, this gives us a bound of $2^t \cdot \ell \cdot \hbar \leqslant 2^t t \ell$ on the total number of vertices in G. If ℓ were small enough, then we might find a solution at this point by brute-force. The rest of this section focuses on addressing the case when ℓ is arbitrary.

To this end, let us call a color q "harmless" if there exists a vertex $v \in T$ for which $c(v) = q$. Note that there are at most t harmless colors. Any other color will be referred to as a "fresh" color in our discussion. We can employ the arguments from the previous paragraph to ensure that the number of vertices in $V \setminus T$ that are colored with the same color are bounded by $2^t h$. Observe that this already gives a bound of $O(2^t t^2)$ on the total number of vertices colored with harmless colors.

We now sketch the ideas for handling the fresh colors, which will complete the description of the algorithm. We partition the vertex set $V \setminus T$ into X and Y, where X contains all vertices of $V \setminus T$ colored with harmless colors, and Y contains all vertices of $V \setminus T$ colored with fresh colors. Define the *signature* of a fresh color q, denoted by $\eth_q$ as a tuple with 2^t coordinates, indexed by subsets of T, where $\eth_q[X]$ is the number of vertices in $\mathcal{J}(X, q)$. Note that the number of signatures is bounded by h^{2^t}.

Let q be a fresh color and let w be the number of vertices of color q. A solution F can be thought of as assigning a label between $1 \leqslant i \leqslant h$ to some collection of $\min\{w, h\}$ vertices of q, indicating which bags the vertices are aligned with in $G \setminus F$. The number of possible assignments is:

$$g(w) := \left(\binom{w}{h} \cdot h! \right).$$

We now describe our process for the fresh colors. Fix a signature $\eth$. Let $C_\eth \subseteq [\ell]$ be the set of all colors that have the same signature, and therefore the same number of vertices (say $w_\eth$). We retain $\min\{C_\eth, g(w)\}$ fresh colors from among those that have signature $\eth$, and assign the others to bags using an optimal strategy. Note that for any particular color class, it is possible to determine the best assignment of the vertices to bags by examining all possible assignments (note that this is FPT in t since the number of bags and the number of vertices of any given color are both bounded).

The correctness of the approach above follows from the fact that vertices with different fresh colors can be assigned independently (as they do not contribute to any conflicts as far as the final components being colorful are concerned). We retain enough vertices to remember all possible ways in which F may have assigned vertices of any particular color. The key observation is that if two colors with the same signature experience the exact same assignment to bags in $G \setminus F$, then one of them may be reassigned to an optimal assignment without affecting the connectivity template of $G \setminus F$. A more formal version of this argument is deferred to the full version of this manuscript. We summarize our discussion in this section in the following claim.

Theorem 3. *CC$[\![\tau]\!]$ admits an* FPT *algorithm.*

6 Concluding Remarks

It would be interesting to see if other structural parameterizations for some of the problems in the colorful components framework become tractable when combined with the number of colors as a parameter. We also believe that the vertex cover parameterization studied in this contribution admits a more efficient algorithm, possibly by establishing an appropriate reduction to the problem of minimizing the number of singleton vertices (MSV), which is known to admit a polynomial time algorithm [2]. The fact that vertices from the independent set either break away into singletons or join the vertex cover vertices motivates this suggestion.

References

1. Adamaszek, A., Blin, G., Popa, A.: Approximation and hardness results for the maximum edges in transitive closure problem. In: Kratochvíl, J., Miller, M., Froncek, D. (eds.) IWOCA 2014. LNCS, vol. 8986, pp. 13–23. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19315-1_2
2. Adamaszek, A., Popa, A.: Algorithmic and hardness results for the colorful components problems. Algorithmica **73**(2), 371–388 (2015)
3. Alon, N., Yuster, R., Zwick, U.: Color-coding. J. Assoc. Comput. Mach. **42**(4), 844–856 (1995)
4. Avidor, A., Langberg, M.: The multi-multiway cut problem. Theor. Comput. Sci **377**(1–3), 35–42 (2007)
5. Bousquet, N., Daligault, J., Thomassé, S.: Multicut is FPT. In: Fortnow, L., Vadhan, S.P. (eds.) Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, pp. 459–468. ACM (2011)
6. Bruckner, S., Hüffner, F., Karp, R.M., Shamir, R., Sharan, R.: Topology-free querying of protein interaction networks. J. Comput. Biol. **17**(3), 237–252 (2010)
7. Bruckner, S., Hüffner, F., Komusiewicz, C., Niedermeier, R.: Evaluation of ILP-based approaches for partitioning into colorful components. In: Bonifaci, V., Demetrescu, C., Marchetti-Spaccamela, A. (eds.) SEA 2013. LNCS, vol. 7933, pp. 176–187. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38527-8_17
8. Bruckner, S., Hüffner, F., Komusiewicz, C., Niedermeier, R., Thiel, S., Uhlmann, J.: Partitioning into colorful components by minimum edge deletions. In: Kärkkäinen, J., Stoye, J. (eds.) CPM 2012. LNCS, vol. 7354, pp. 56–69. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31265-6_5
9. Corel, E., Pitschi, F., Morgenstern, B.: A min-cut algorithm for the consistency problem in multiple sequence alignment. Bioinformatics **26**(8), 1015–1021 (2010)
10. 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
11. Diestel, R.: Graph Theory. Springer Graduate Text GTM 173. Reinhard Diestel, July 2012

12. Dondi, R., Sikora, F.: Parameterized complexity and approximation issues for the colorful components problems. In: Beckmann, A., Bienvenu, L., Jonoska, N. (eds.) CiE 2016. LNCS, vol. 9709, pp. 261–270. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40189-8_27
13. Fellows, M.R., Fertin, G., Hermelin, D., Vialette, S.: Upper and lower bounds for finding connected motifs in vertex-colored graphs. J. Comput. Syst. Sci. **77**(4), 799–811 (2011)
14. Guillemot, S., Sikora, F.: Finding and counting vertex-colored subtrees. Algorithmica **65**(4), 828–844 (2013)
15. Marx, D., Razgon, I.: Fixed-parameter tractability of multicut parameterized by the size of the cutset. SIAM J. Comput. **43**(2), 355–388 (2014)
16. Zheng, C., Swenson, K., Lyons, E., Sankoff, D.: OMG! Orthologs in multiple genomes – competing graph-theoretical formulations. In: Przytycka, T.M., Sagot, M.-F. (eds.) WABI 2011. LNCS, vol. 6833, pp. 364–375. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23038-7_30

Analysis of Information Leakage Due to Operative Errors in Card-Based Protocols

Takaaki Mizuki[1(✉)] and Yuichi Komano[2]

[1] Tohoku University, Sendai, Japan
tm-paper+cardecc@g-mail.tohoku-university.jp
[2] Toshiba Corporation, Kawasaki, Japan
yuichi1.komano@toshiba.co.jp

Abstract. Card-based protocols provide secure multi-party computation using a deck of physical cards, via a series of operations such as shuffling and turning over cards, which are supposed to be executed by humans. Although almost all existing protocols have been designed to be perfectly secure, operative errors by humans would cause information leakage. In this paper, we propose a technique for analyzing information leakage due to operative errors in card-based protocols. To be specific, we introduce a concept, which we call a "probability trace," and propose a new diagram for displaying how much information leaks, by enhancing the KWH diagram proposed by Koch, Walzer, and Härtel. By applying our diagram to a card-based protocol with an operative error, we can precisely reveal the leakage of players' inputs from the protocol output. We also discuss an application of the diagram to human-error-free implementations of the existing six-card AND protocol.

1 Introduction

Card-based protocols allow us to securely perform computations, such as AND and OR, among players with a physical deck of playing cards, as depicted in Fig. 1. Because their operations are easy to implement and both their correctness and security are intuitively understandable, card-based protocols have been widely used to solve social problems in daily life as well as to educate non-experts about cryptography [1]. Many protocols have been proposed [2]. Let us review a seminal work, the six-card (committed-format) AND protocol [3], which is one of the most *practical* protocols, and has been applied as a learning tool thanks to its simple operations.

In this protocol, a one-bit value is encoded with two cards, black $\boxed{\clubsuit}$ and red $\boxed{\heartsuit}$, as:

$$\boxed{\clubsuit}\boxed{\heartsuit} = 0 \text{ and } \boxed{\heartsuit}\boxed{\clubsuit} = 1.$$

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 250–262, 2018.
https://doi.org/10.1007/978-3-319-94667-2_21

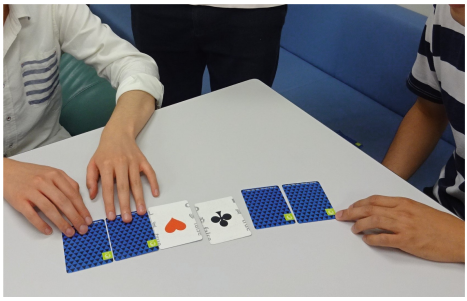
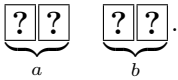


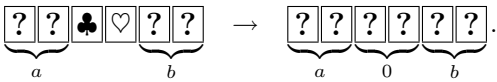
Fig. 1. Execution of a card-based protocol using physical playing cards

Using this encoding rule, two players, Alice and Bob, place two face-down cards depending on their secret one-bit inputs a and b, respectively, as:

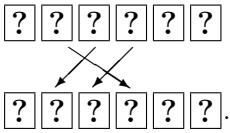


The left and right pairs of two cards are called *commitments* to $a \in \{0, 1\}$ and $b \in \{0, 1\}$, respectively. With these commitments and two additional cards, the six-card AND protocol consists of the following five steps:

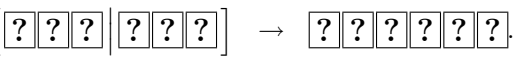
1. Place black and red cards (which become a commitment to 0) between the commitments to a and b, and turn the cards face down:



2. Rearrange the cards as:

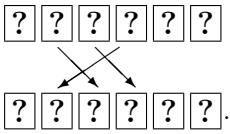


3. Divide the sequence of cards into two halves and randomly shuffle them keeping the order of cards inside the half portion unchanged:

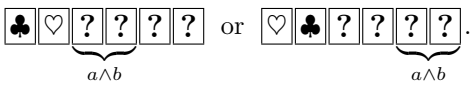


   This operation is called a *random bisection cut* and it is securely implementable by humans [4].

4. Rearrange the cards as:



5. Open the leftmost two cards and obtain a commitment to $a \wedge b$ as:



The commitment to $a \wedge b$ remains face down, and hence, the players can use it as an input to a subsequent execution of the six-card AND protocol. By repeating the protocol, more than two players can securely compute the AND of their input bits.

The above six-card protocol is simple, and therefore, from our educational experiments with non-experts, people can almost always execute the protocol without any mistakes. In rare instances, however, operative errors such as confusing the rearrangement operations in Steps 2 and 4 occur.

In this paper, we propose a technique to analyze the information leakage caused by such operative errors in card-based protocols. To be specific, we introduce a concept, which we call a *probability trace*, and propose a new diagram, which employs probability traces, for the leakage analysis. Our diagram is an extension of the KWH diagram invented by Koch et al. [5]. Similar to the KWH diagram, our diagram can be applied to analyze the correctness and security of a card-based protocol without operative errors. In addition, by applying our diagram to a card-based protocol with an operative error, we can analyze the leakage of players' inputs from the protocol output. We then discuss an application of the diagram to human-error-free implementations of the six-card AND protocol above. That is, we discuss solutions to the problem of detecting the operative errors and preventing information leakage in the AND protocol.

The remainder of this paper is organized as follows. Section 2 reviews the background information on the card-based protocol. We then introduce the concepts of a probability trace and the extended diagram in Sect. 3. In Sect. 4, we discuss the erroneous protocols with operative errors as examples, and the application of our diagram to analyzing their correctness and security. Sections 5 and 6 discuss the detection of the operative errors and protection against them, respectively. Finally, Sect. 7 concludes this paper.

2 Background

Let us review the formal definitions of card-based protocols and consider an example of a card-based protocol. In this paper, $|S|$ denotes the cardinal number of set S.

2.1 Definitions of Card-Based Protocols

The computational model of card-based protocols was introduced by Mizuki and Shizuya [6], after which Koch et al. [5] and Mizuki and Shizuya [2] refined it. Let us review the definition as follows.

In this paper, we deal with a card-based protocol using two *atomic symbols*, ♣ and ♡. A non-empty finite multi-set $\mathcal{D}$ of atomic symbols is called a *deck*.

An example of a deck for the six-card AND protocol is $[\clubsuit, \clubsuit, \clubsuit, \heartsuit, \heartsuit, \heartsuit]$. The symbol on the back of a card is denoted by "?".

A card, such as $\boxed{\clubsuit}$ and $\boxed{?}$, which is placed on a table is called a *lying card*. Formally, face-up and face-down lying cards for $c \in \mathcal{D}$ are denoted by $\frac{c}{?}$ and $\frac{?}{c}$, respectively.

The input and output of a card-based protocol are sequences of lying cards. For example, the input to the six-card AND protocol is a sequence of six cards placed on the table, where $a, b \in \{0, 1\}$:

$$\underbrace{\boxed{?}\boxed{?}}_{a}\boxed{\clubsuit}\boxed{\heartsuit}\underbrace{\boxed{?}\boxed{?}}_{b}.$$

Therefore, there are four possible initial sequences Γ_0^{ab}:

$$\Gamma_0^{00} = \left(\frac{?}{\clubsuit}, \frac{?}{\heartsuit}, \frac{\clubsuit}{?}, \frac{\heartsuit}{?}, \frac{?}{\clubsuit}, \frac{?}{\heartsuit}\right),\ \Gamma_0^{01} = \left(\frac{?}{\clubsuit}, \frac{?}{\heartsuit}, \frac{\clubsuit}{?}, \frac{\heartsuit}{?}, \frac{?}{\heartsuit}, \frac{?}{\clubsuit}\right),$$

$$\Gamma_0^{10} = \left(\frac{?}{\heartsuit}, \frac{?}{\clubsuit}, \frac{\clubsuit}{?}, \frac{\heartsuit}{?}, \frac{?}{\clubsuit}, \frac{?}{\heartsuit}\right),\ \text{and}\ \Gamma_0^{11} = \left(\frac{?}{\heartsuit}, \frac{?}{\clubsuit}, \frac{\clubsuit}{?}, \frac{\heartsuit}{?}, \frac{?}{\heartsuit}, \frac{?}{\clubsuit}\right).$$

The set of all sequences of lying cards is denoted by

$$\mathsf{Seq}^{\mathcal{D}} \stackrel{\text{def}}{=} \{\Gamma \mid \Gamma \text{ is a sequence of lying cards for } \mathcal{D}\}.$$

For a sequence of lying cards, we call a sequence of their atomic symbols an *atomic sequence*. For example, there are four atomic sequences, each of which corresponds to an input Γ_0^{ab} of the six-card AND protocol:

$$\Gamma_0^{00} : \clubsuit\heartsuit\clubsuit\heartsuit\clubsuit\heartsuit,\ \ \Gamma_0^{01} : \clubsuit\heartsuit\clubsuit\heartsuit\heartsuit\clubsuit,\ \ \Gamma_0^{10} : \heartsuit\clubsuit\clubsuit\heartsuit\clubsuit\heartsuit,\ \ \Gamma_0^{11} : \heartsuit\clubsuit\clubsuit\heartsuit\heartsuit\clubsuit$$

Given a lying card $\frac{c}{?}$ or $\frac{?}{c}$, top returns its visible symbol; i.e., $\mathsf{top}(\frac{c}{?}) = c$, and $\mathsf{top}(\frac{?}{c}) = ?$. For a sequence of d lying cards $\Gamma = (\alpha_1, \alpha_2, \ldots, \alpha_d)$, we call $\mathsf{top}(\Gamma) = (\mathsf{top}(\alpha_1), \mathsf{top}(\alpha_2), \ldots, \mathsf{top}(\alpha_d))$ a *visible sequence* of Γ. We also define the *visible sequence set* $\mathsf{Vis}^{\mathcal{D}}$ of $\mathcal{D}$ as

$$\mathsf{Vis}^{\mathcal{D}} \stackrel{\text{def}}{=} \{\mathsf{top}(\Gamma) \mid \Gamma \in \mathsf{Seq}^{\mathcal{D}}\}.$$

The definitions of card-based protocols are given as follows.

Definition 1 (Card-based protocol [2]). *A* card-based protocol *is specified with a quadruple* $\mathcal{P} = (\mathcal{D}, U, Q, A)$*:*

- $\mathcal{D}$ *is a deck;*
- $U \subseteq \mathsf{Seq}^{\mathcal{D}}$ *is an* input set*;*
- Q *is a* state set *having an* initial state $q_0 \in Q$ *and a* final state $q_{\mathrm{f}} \in Q$*;*
- $A : (Q - \{q_{\mathrm{f}}\}) \times \mathsf{Vis}^{\mathcal{D}} \to Q \times \mathsf{Action}$ *is an* action function*, where* Action *is the set of the following actions:*

- $(\mathsf{turn}, T \subseteq \{1, 2, \ldots, |\mathcal{D}|\})$*; turning over the* i*-th card for each* $i \in T$*,*
- $(\mathsf{perm}, \pi \in S_{|\mathcal{D}|})$*; applying the permutation* π *to a sequence of lying cards where* S_i *denotes the symmetric group of degree* i*,*
- $(\mathsf{shuf}, \Pi \subseteq S_{|\mathcal{D}|}, \mathcal{F})$*; applying* π*, drawn from* Π *according to the probability distribution*[1] $\mathcal{F}$*, to a sequence of lying cards,*
- $(\mathsf{result}, p_1, \ldots, p_\ell)$*; assigning the positions of output commitments with* $p_i \in \{1, 2, \ldots, |\mathcal{D}|\}$*.*

The protocol is correct *if it produces the correct output at the final state.*

The protocol $\mathcal{P} = (\mathcal{D}, U, Q, A)$ proceeds as the Turing machine does. That is, starting from the initial state q_0 and the initial sequence $\Gamma_0 \in U$, its current state q and sequence Γ move to the next state q' and sequence Γ', respectively, according to the output of the action function A.

Definition 2 (Perfect security of card-based protocol). *We call a card-based protocol* $\mathcal{P}$ perfectly secure *if it leaks no information for any run of the protocol (in other words, the input and the visible sequence trace are independent).*

2.2 Six-Card AND Protocol

The six-card AND protocol [3], explained in Sect. 1, can be described as follows. The following sections review this protocol as an example.

Six-card AND protocol $\mathcal{P}^{\mathsf{MS}}$
Input set:

$$\left\{ \Gamma_0^{00} = \left(\frac{?}{\clubsuit}, \frac{?}{\heartsuit}, \frac{\clubsuit}{?}, \frac{\heartsuit}{?}, \frac{?}{\clubsuit}, \frac{?}{\heartsuit}\right), \Gamma_0^{01} = \left(\frac{?}{\clubsuit}, \frac{?}{\heartsuit}, \frac{\clubsuit}{?}, \frac{\heartsuit}{?}, \frac{?}{\heartsuit}, \frac{?}{\clubsuit}\right), \right.$$
$$\left. \Gamma_0^{10} = \left(\frac{?}{\heartsuit}, \frac{?}{\clubsuit}, \frac{\clubsuit}{?}, \frac{\heartsuit}{?}, \frac{?}{\clubsuit}, \frac{?}{\heartsuit}\right), \Gamma_0^{11} = \left(\frac{?}{\heartsuit}, \frac{?}{\clubsuit}, \frac{\clubsuit}{?}, \frac{\heartsuit}{?}, \frac{?}{\heartsuit}, \frac{?}{\clubsuit}\right) \right\}$$

Steps:

1. $(\mathsf{turn}, \{3, 4\})$
2. $(\mathsf{perm}, (2\,4\,3))$
3. $(\mathsf{shuf}, \{\mathsf{id}, (1\,4)(2\,5)(3\,6)\})$
4. $(\mathsf{perm}, (2\,3\,4))$
5. $(\mathsf{turn}, \{1, 2\})$
   if visible seq. $= (\clubsuit, \heartsuit, ?, ?, ?, ?)$ **then** $(\mathsf{result}, 3, 4)$
   else if visible seq. $= (\heartsuit, \clubsuit, ?, ?, ?, ?)$ **then** $(\mathsf{result}, 5, 6)$

---

[1] We omit the description of $\mathcal{F}$ if the distribution is uniform.

3 Probability Trace and the Extended KWH Diagram

Koch et al. [5] introduced a diagram (the KWH diagram) to check the correctness and security of card-based protocols. In this section, we introduce a new concept, which we call a *probability trace*, and a new diagram created by enhancing the KWH diagram with the probability trace. We first show an example of our diagram, after which we give a formal definition of the probability trace.

3.1 Example of Our Diagram

In this subsection, we give an example of our diagram which shows the correctness and perfect security of the six-card AND protocol $\mathcal{P}^{\mathsf{MS}}$. Let p_{ab} be the probabilities that a and b are private inputs of Alice and Bob, respectively. The first component of our diagram, which displays the status after Step 1, is:

$$\begin{aligned}
\clubsuit\heartsuit\clubsuit\heartsuit\clubsuit\heartsuit &: (p_{00}, 0, 0, 0)\\
\clubsuit\heartsuit\clubsuit\heartsuit\heartsuit\clubsuit &: (0, p_{01}, 0, 0)\\
\heartsuit\clubsuit\clubsuit\heartsuit\clubsuit\heartsuit &: (0, 0, p_{10}, 0)\\
\heartsuit\clubsuit\clubsuit\heartsuit\heartsuit\clubsuit &: (0, 0, 0, p_{11}).
\end{aligned}$$

Each line consists of an atomic sequence and the corresponding "probability trace" (which will be defined formally in the next subsection). The probability trace consists of four probabilities corresponding to Γ_0^{00}, Γ_0^{01}, Γ_0^{10}, and Γ_0^{11}; for example, the atomic sequence $\clubsuit\heartsuit\clubsuit\heartsuit\clubsuit\heartsuit$ is derived only when the input sequence is Γ_0^{00} with probability of p_{00}, and hence, the first coordinate of the probability trace is p_{00} and the remaining three coordinates are all 0.

Figure 2 depicts the whole diagram with probability traces for the protocol $\mathcal{P}^{\mathsf{MS}}$. In this figure, we denote the end of Step i by "Step i" for brevity. We omit the component for input, i.e., the initial sequence, because it is the same as that of Step 1 explained above except the visible sequence.

The component for Step 2 is derived from the rearrangement action, and its probability traces are unchanged from the previous step. The same is true for Step 4. The component for Step 3 originates from the bisection cut. It is the sum of the following two possible outcomes (left and right below) from the bisection cut:

$$\begin{aligned}
\clubsuit\clubsuit\heartsuit\heartsuit\clubsuit\heartsuit &: (p_{00}/2, 0, 0, 0) & \heartsuit\clubsuit\heartsuit\clubsuit\clubsuit\heartsuit &: (p_{00}/2, 0, 0, 0)\\
\clubsuit\clubsuit\heartsuit\heartsuit\heartsuit\clubsuit &: (0, p_{01}/2, 0, 0) & \heartsuit\heartsuit\clubsuit\clubsuit\clubsuit\heartsuit &: (0, p_{01}/2, 0, 0)\\
\heartsuit\clubsuit\heartsuit\clubsuit\clubsuit\heartsuit &: (0, 0, p_{10}/2, 0) & \clubsuit\clubsuit\heartsuit\heartsuit\clubsuit\heartsuit &: (0, 0, p_{10}/2, 0)\\
\heartsuit\clubsuit\heartsuit\clubsuit\heartsuit\clubsuit &: (0, 0, 0, p_{11}/2) & \clubsuit\heartsuit\clubsuit\heartsuit\clubsuit\heartsuit &: (0, 0, 0, p_{11}/2).
\end{aligned}$$

The component for Step 5 is from the turn action for the leftmost two cards. Note that there are two visible sequences, $\clubsuit\heartsuit????$ and $\heartsuit\clubsuit????$.

The positions of the output commitment are determined by turning over the leftmost two cards at Step 5. If these cards are $\clubsuit\heartsuit$ (resp. $\heartsuit\clubsuit$), they are the

Step	Visible sequence	Atomic sequence	Probability trace
1	? ? ? ? ? ?	♣♡♣♡♣♡	$(p_{00}, 0, 0, 0)$
		♣♡♣♡♡♣	$(0, p_{01}, 0, 0)$
		♡♣♣♡♣♡	$(0, 0, p_{10}, 0)$
		♡♣♣♡♡♣	$(0, 0, 0, p_{11})$
2	? ? ? ? ? ?	♣♣♡♡♣♡	$(p_{00}, 0, 0, 0)$
		♣♣♡♡♡♣	$(0, p_{01}, 0, 0)$
		♡♣♡♣♣♡	$(0, 0, p_{10}, 0)$
		♡♣♡♣♡♣	$(0, 0, 0, p_{11})$
3	? ? ? ? ? ?	♣♣♡♡♣♡	$(p_{00}/2, 0, p_{10}/2, 0)$
		♣♣♡♡♡♣	$(0, p_{01}/2, 0, 0)$
		♣♡♣♡♣♡	$(0, 0, 0, p_{11}/2)$
		♡♣♡♣♣♡	$(p_{00}/2, 0, p_{10}/2, 0)$
		♡♡♣♣♣♡	$(0, p_{01}/2, 0, 0)$
		♡♣♡♣♡♣	$(0, 0, 0, p_{11}/2)$
4	? ? ? ? ? ?	♣♡♣♡♣♡	$(p_{00}/2, 0, p_{10}/2, 0)$
		♣♡♣♡♡♣	$(0, p_{01}/2, 0, 0)$
		♣♡♡♣♣♡	$(0, 0, 0, p_{11}/2)$
		♡♣♣♡♣♡	$(p_{00}/2, 0, p_{10}/2, 0)$
		♡♣♡♣♣♡	$(0, p_{01}/2, 0, 0)$
		♡♣♣♡♡♣	$(0, 0, 0, p_{11}/2)$
5	♣♡? ? ? ?	♣♡♣♡♣♡	$(p_{00}, 0, p_{10}, 0)$
		♣♡♣♡♡♣	$(0, p_{01}, 0, 0)$
		♣♡♡♣♣♡	$(0, 0, 0, p_{11})$
	♡♣? ? ? ?	♡♣♣♡♣♡	$(p_{00}, 0, p_{10}, 0)$
		♡♣♡♣♣♡	$(0, p_{01}, 0, 0)$
		♡♣♣♡♡♣	$(0, 0, 0, p_{11})$

Fig. 2. Our diagram for the six-card AND protocol

middle (resp. rightmost) two cards, indicated with underlines at Step 5 in Fig. 2. The correctness of the protocol $\mathcal{P}^{\mathsf{MS}}$ is easily checked by comparing $i \wedge j$ for non-zero probability p_{ij} in each probability trace with the corresponding underlined output.

We then discuss the security of the protocol $\mathcal{P}^{\mathsf{MS}}$. For the case where the leftmost two cards are ♣♡ in Fig. 2, the sum of the probability traces is $(p_{00}, p_{01}, p_{10}, p_{11})$. That is, the probability distribution of the input after the leftmost two cards are opened is unchanged from the viewpoint of the players and others. This means that no information leaks through the protocol. Similarly, no information leaks for the case where the leftmost two cards are ♡♣. Hence, we have confirmed the perfect security of the protocol.

3.2 Definition of Probability Trace

We present a formal definition of a probability trace as follows.

Definition 3 (Probability trace). *Let $n = |U|$ for input set U of a card-based protocol $\mathcal{P}$. An n-tuple $(q_{1,j}, \cdots, q_{n,j})$ such that*

$$q_{i,j} = \Pr[M = \Gamma_0^i, G_j = s | V_j = v]$$

is called a probability trace *for a step number j, an atomic sequence s, and a visible sequence trace v, where M, G_j, and V_j are random variables of the original input sequence, of the atomic sequence for the end of the j-th step, and of the visible sequence trace for the end of the j-th step, respectively.*

Note that the probability traces in Fig. 2 are actually obtained according to Definition 3. The original KWH diagram is a sequence of pairs of an atomic sequence and its probability. We replace the probability with the probability trace. Our diagram allows us to analyze the leakage from operative errors in the protocol (as will be seen in the next section), in addition to checking the correctness and security of the protocol as the original KWH diagram does.

4 Rearrangement Errors in Six-Card and Protocol

In this section, we apply our diagram to analyze the information leakage from operative errors in the six-card AND protocol $\mathcal{P}^{\mathsf{MS}}$.

4.1 Classification of Rearrangement Error

We discuss operative errors in $\mathcal{P}^{\mathsf{MS}}$. We assume that two players, Alice and Bob, perform wrong rearrangements by mistake at Steps 2 and 4 in $\mathcal{P}^{\mathsf{MS}}$, and they are not aware of this. There are three error types, in addition to the correct rearrangements; we name such erroneous protocols $\mathcal{P}_1, \mathcal{P}_2$, and $\mathcal{P}_3$, as follows.

- Protocol $\mathcal{P}^{\mathsf{MS}}$: $(\mathsf{perm}, (2\,4\,3))$ in Step 2, and $(\mathsf{perm}, (2\,3\,4))$ in Step 4
- Protocol $\mathcal{P}_1$: $(\mathsf{perm}, (2\,3\,4))$ in Step 2, and $(\mathsf{perm}, (2\,4\,3))$ in Step 4 (erroneous)
- Protocol $\mathcal{P}_2$: $(\mathsf{perm}, (2\,3\,4))$ in Step 2, and $(\mathsf{perm}, (2\,3\,4))$ in Step 4 (erroneous)
- Protocol $\mathcal{P}_3$: $(\mathsf{perm}, (2\,4\,3))$ in Step 2, and $(\mathsf{perm}, (2\,4\,3))$ in Step 4 (erroneous)

In a similar way to the correct protocol $\mathcal{P}^{\mathsf{MS}}$ in Sect. 3, diagrams for erroneous protocols can be described. Figure 3 depicts the final part of the diagram for $\mathcal{P}_1$. Diagrams for $\mathcal{P}_2$ and $\mathcal{P}_3$ can be depicted; however, due to the limitation of space, we omit them.

As known from Fig. 3, there are four classes of protocol output:

(a) A correct commitment to $a \wedge b$
(b) A complementary commitment to $\overline{a \wedge b}$
(c) An invalid commitment, such as ♣♣ or ♡♡
(d) No commitment because the leftmost two cards are invalid (♣♣ or ♡♡)

Step	Visible sequence	Atomic sequence	Output	Probability trace
5	♣♡?????	♣♡♣♡♣♡	♣♡	$(\frac{p_{00}}{2p_{00}+p_{01}+p_{10}}, 0, 0, 0)$
		♣♡♣♡♡♣	♣♡	$(0, \frac{p_{01}}{2p_{00}+p_{01}+p_{10}}, 0, 0)$
		♣♡♣♣♡♡	♣♣	$(\frac{p_{00}}{2p_{00}+p_{01}+p_{10}}, 0, 0, 0)$
		♣♡♡♣♡♣	♡♣	$(0, 0, \frac{p_{10}}{2p_{00}+p_{01}+p_{10}}, 0)$
	♡♣????	♡♣♣♡♣♡	♣♡	$(0, 0, \frac{p_{10}}{p_{10}+p_{11}}, 0)$
		♡♣♣♡♡♣	♡♣	$(0, 0, 0, \frac{p_{11}}{p_{10}+p_{11}})$
	♣♣????	♣♣♣♡♡♡	–	$(0, \frac{p_{01}}{p_{01}+p_{11}}, 0, 0)$
		♣♣♡♡♡♣	–	$(0, 0, 0, \frac{p_{11}}{p_{01}+p_{11}})$

Fig. 3. Final part of our diagram for $\mathcal{P}_1$

Table 1. Output classes for each protocol

Protocol	(a)	(b)	(c)	(d)
$\mathcal{P}^{\mathsf{MS}}$	8	0	0	0
$\mathcal{P}_1$	4	1	1	2
$\mathcal{P}_2$	4	0	0	4
$\mathcal{P}_3$	1	3	0	4

From (a) to (c), the leftmost two cards are either ♣♡ or ♡♣, and the positions of the commitment are determined in the protocol; in contrast, in case (d), the leftmost two cards are invalid in the protocol, and therefore, the players can become aware of their operative error. Table 1 summarizes the output classes derived from the correct protocol in Fig. 2 and the erroneous protocols in Fig. 3 and other diagrams.

4.2 Correctness and Security of Erroneous Protocols

We have already discussed the correctness and security of the correct protocol ($\mathcal{P}^{\mathsf{MS}}$) in Sect. 3.1. Let us now discuss the correctness and security of the erroneous protocols ($\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$) with Table 1.

For $\mathcal{P}_1$, half of the eight cases lead to correct outputs; however, the remaining four lead to incorrect/no outputs. Note that the output class (c) appears only in $\mathcal{P}_1$. If an output is in class (d), that is, if the leftmost two cards are ♣♣, the probability traces tell us that the input pair (a, b) must be either $(0, 1)$ or $(1, 1)$; that is, $b = 1$ leaks. If the leftmost two cards are ♣♡, information about the input pair leaks for a person who knows that the erroneous protocol $\mathcal{P}_1$ was executed, as follows. Let us consider the sum of the probability traces of the first four lines in Fig. 3; then, we have

$$\left(\frac{2p_{00}}{2p_{00}+p_{01}+p_{10}}, \frac{p_{01}}{2p_{00}+p_{01}+p_{10}}, \frac{p_{10}}{2p_{00}+p_{01}+p_{10}}, 0\right).$$

It is known from the coordinates that $(a, b) \neq (1, 1)$ leaks. More precisely, the above four-tuple is the distribution on (a, b) after $\mathcal{P}_1$ terminates with ♣♡. Hence, $\mathcal{P}_1$ is not perfectly secure.

Similar to the analysis for $\mathcal{P}_1$, we can show that the protocols $\mathcal{P}_2$ and $\mathcal{P}_3$ are neither correct nor perfectly secure with our diagrams.

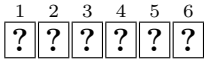
Thus, our new diagram with probability traces displays exactly how much information leaks.

5 Detection of Operative Errors in Card Rearrangement

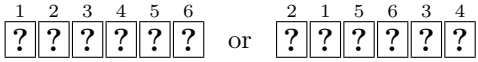
As discussed above, if the atomic symbols of the leftmost two cards are the same, players can become aware of their operative error. In this section, we propose two other methods to detect operative errors as in $\mathcal{P}_1, \mathcal{P}_2$, and $\mathcal{P}_3$.

5.1 Detection with Card Arrangement

For the sake of convenience, let us assign a label for the six cards after Step 1.



After Step 4, the order of these cards is one of the following with probability $\frac{1}{2}$:



Note that, in either case, the leftmost two cards correspond to the input a where their positions are randomly switched by the bisection cut. The positions of pairs of the middle two cards and the rightmost two cards are randomly switched, whereas the orders of the two cards in each pair are unchanged.

Assume that the back of the card is asymmetric.[2] Let us place the leftmost two cards upside down at Step 1:

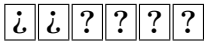


Table 2 summarizes the visible sequences after Step 4 in each protocol. From the visible sequence, players can check whether the protocol has been correctly executed or not (except for $\mathcal{P}_1$ with probability $\frac{1}{2}$); if not, they can determine which type of operative error has occurred. Note that the exception can be handled: repeating Steps from 2 to 4 is effective in decreasing the error.

[2] If the back of the card is symmetric, it is possible to make the back asymmetric by putting a mark on it.

Table 2. Visible sequences after Step 4 in each protocol

Protocol	Visible sequence
$\mathcal{P}^{\mathsf{MS}}$	¿ ¿ ? ? ? ? or ¿ ¿ ? ? ? ?
$\mathcal{P}_1$	¿ ¿ ? ? ? ? or ? ? ¿ ? ? ¿
$\mathcal{P}_2$	¿ ? ¿ ? ? ? or ¿ ? ¿ ? ? ?
$\mathcal{P}_3$	¿ ? ? ¿ ? ? or ? ¿ ? ? ? ¿

5.2 Detection with Discarded Cards

In $\mathcal{P}^{\mathsf{MS}}$, the leftmost two cards are opened at the final step, the middle two cards or the rightmost two cards are selected as a commitment to the result, and the remaining two cards are discarded. With these discarded cards, we can detect an error leading to an output in class (c). To be specific, players randomly shuffle these cards and then open them. Players can become aware of such an error if the revealed two cards have the same color.

6 Toward Human-Error-Free Protocol on Rearrangement

We propose two methods to avoid the arrangement errors in $\mathcal{P}_1, \mathcal{P}_2$, and $\mathcal{P}_3$.

6.1 Card Arrangement Without Rearrangement

Kastner et al. [7] described a two-dimensional arrangement for the protocol $\mathcal{P}^{\mathsf{MS}}$ as in Fig. 4. Two cards for input a are placed at the first row across the dashed line. Two cards for input b are placed at the left side of the second row; and two cards $\boxed{\clubsuit}\boxed{\heartsuit}$ are placed at the right side. Figure 4 above shows the face-up cards for $\Gamma_0^{00}, \Gamma_0^{01}, \Gamma_0^{10}$, and Γ_0^{11}, from left to right.

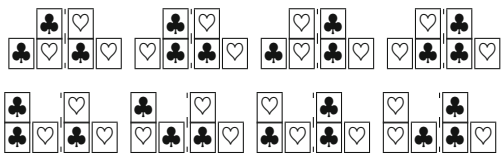


Fig. 4. Arrangements in Kastner et al. (above) and our modification (below)

In their arrangement, tuples of three face-down cards across the dashed line are randomly shuffled (bisection cut), after which two cards in the first row are opened. If they are $\boxed{\clubsuit}\boxed{\heartsuit}$ (resp. $\boxed{\heartsuit}\boxed{\clubsuit}$), then the rightmost (resp. leftmost) two cards are a commitment to the result.

Although their implementation is essentially the same as $\mathcal{P}^{\mathsf{MS}}$, it requires no rearrangement of cards. However, before and after the shuffle, three cards should be layered, which might lead to another operative error.

Let us consider another arrangement, the one depicted in Fig. 4 below. The operation is the same as that of Kastner et al. The arrangements of the three cards across the dashed line are identical; therefore, we expect to decrease the probability of operative errors in layering the cards.

6.2 Using Mixed Encoding Rules

The protocol $\mathcal{P}^{\mathsf{MS}}$ uses an encoding where $0 = \clubsuit\heartsuit$ and $1 = \heartsuit\clubsuit$ with two cards each. Mizuki [8] introduced another four-card AND protocol $\mathcal{P}^{\mathsf{M}}$ using two different encodings. In addition to the above encoding, $\mathcal{P}^{\mathsf{M}}$ also uses an encoding with one card: $0 = \clubsuit$ and $1 = \heartsuit$. In $\mathcal{P}^{\mathsf{M}}$, each input is encoded with a different rule.

$$\boxed{\clubsuit}\underbrace{\boxed{?}}_{a}\underbrace{\boxed{?}\boxed{?}}_{b}.$$

$\mathcal{P}^{\mathsf{M}}$ requires no rearrangement, and therefore, it can avoid an operative error in rearrangement. One possible drawback is that the complicated encoding rule might lead to another error in the arrangement.

7 Conclusions

In this paper, we introduced the concept of a probability trace and a diagram with the probability trace, for analyzing information leakage due to operative errors in a card-based protocol. We then precisely showed, with our diagram, how much information leaks from an operative error in $\mathcal{P}^{\mathsf{MS}}$. We also discussed the detection of operative errors and protection against them.

Acknowledgments. This work was supported by JSPS KAKENHI Grant Number JP17K00001.

References

1. Marcedone, A., Wen, Z., Shi, E.: Secure dating with four or fewer cards. Cryptology ePrint Archive, Report 2015/1031 (2015)
2. Mizuki, T., Shizuya, H.: Computational model of card-based cryptographic protocols and its applications. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **E100.A**(1), 3–11 (2017)
3. Mizuki, T., Sone, H.: Six-card secure AND and four-card secure XOR. In: Deng, X., Hopcroft, J.E., Xue, J. (eds.) FAW 2009. LNCS, vol. 5598, pp. 358–369. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02270-8_36
4. Ueda, I., Nishimura, A., Hayashi, Y., Mizuki, T., Sone, H.: How to implement a random bisection cut. In: Martín-Vide, C., Mizuki, T., Vega-Rodríguez, M.A. (eds.) TPNC 2016. LNCS, vol. 10071, pp. 58–69. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49001-4_5

5. Koch, A., Walzer, S., Härtel, K.: Card-based cryptographic protocols using a minimal number of cards. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, Part I. LNCS, vol. 9452, pp. 783–807. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48797-6_32
6. Mizuki, T., Shizuya, H.: A formalization of card-based cryptographic protocols via abstract machine. Int. J. Inf. Secur. **13**(1), 15–23 (2014)
7. Kastner, J., Koch, A., Walzer, S., Miyahara, D., Hayashi, Y., Mizuki, T., Sone, H.: The minimum number of cards in practical card-based protocols. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part III. LNCS, vol. 10626, pp. 126–155. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70700-6_5
8. Mizuki, T.: Card-based protocols for securely computing the conjunction of multiple variables. Theor. Comput. Sci. **622**, 34–44 (2016)

Zero-Suppression and Computation Models

Hiroki Morizumi(✉)

Interdisciplinary Graduate School of Science and Engineering, Shimane University, Matsue, Shimane 690-8504, Japan
morizumi@cis.shimane-u.ac.jp

Abstract. Zero-suppressed binary decision diagrams (ZDDs) are a data structure representing Boolean functions, and one of the most successful variants of binary decision diagrams (BDDs). On the other hand, BDDs are also called branching programs in computational complexity theory, and have been studied as a computation model. In this paper, we consider ZDDs from the viewpoint of computational complexity theory. Firstly, we define zero-suppressed branching programs, which actually have the same definition to (unordered) ZDDs, and consider the computational power of zero-suppressed branching programs. Secondly, we attempt to generalize the concept of zero-suppression. We call the basic idea of ZDDs zero-suppression. We show that zero-suppression can be applied to other two classical computation models, decision trees and Boolean formulas.

1 Introduction

Zero-suppressed binary decision diagrams (ZDDs) are a data structure representing Boolean functions, introduced by Minato [6], and one of the most successful variants of binary decision diagrams (BDDs). Knuth has referred to ZDDs as an important variant of BDDs in his book [4], and ZDDs are also referred to in other books [5,7]. On the other hand, BDDs are also called branching programs in computational complexity theory, and have been studied as a computation model.

ZDDs have the same shape as BDDs (and branching programs) have, and the only difference is the way to determine the output. An assignment to the variables determines a computation path from the start node to a sink node. ZDDs output 1 iff the value of the sink node is 1 and all variables which are not contained in the computation path are assigned by 0. (See Sect. 2 for the formal definitions.) ZDDs are known to be effective in representing a certain kind of Boolean functions such as almost all outputs are 0. (Thus, ZDDs are better for Boolean functions characterizing sparse sets of combinations than BDDs.) In this paper, we consider ZDDs from the viewpoint of computational complexity theory.

H. Morizumi—This work was supported by JSPS KAKENHI Grant Number 15K11986.

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 263–272, 2018.
https://doi.org/10.1007/978-3-319-94667-2_22

In the first part of this paper, we consider ZDDs and branching programs. Branching programs are known as a computation model to approach the L vs. P problem. It is known that the class of decision problems solvable by a nonuniform family of polynomial-size branching programs is equal to $\mathsf{L/poly}$ [3]. $\mathsf{L/poly}$ is the class of decision problems solvable by nonuniform logarithmic space Turing machines. If one have proven a superpolynomial lower bound for the size of branching programs computing a Boolean function in P, then $\mathsf{L} \neq \mathsf{P}$. In this paper, we define zero-suppressed branching programs, which actually have the same definition to (unordered) ZDDs, and consider the following question: Is the class of decision problems solvable by a nonuniform family of polynomial-size zero-suppressed branching programs equal to $\mathsf{L/poly}$? We prove three results which are related to the question.

Our three results for zero-suppressed branching programs are summarized as follows. Firstly, we prove that the class of decision problems solvable by a nonuniform family of polynomial-size width 5 (or arbitrary constant which is greater than 5) zero-suppressed branching programs is equal to nonuniform NC^1. This corresponds to the well-known Barrington's theorem [1], which showed that the class of decision problems solvable by a nonuniform family of polynomial-size width 5 branching programs is equal to nonuniform NC^1. Secondly, we prove that the class of decision problems solvable by a nonuniform family of polynomial-size zero-suppressed branching programs contains $\mathsf{L/poly}$, and is contained in nonuniform NC^2. Thirdly, we prove that the class of decision problems solvable by a nonuniform family of polynomial-size read-once zero-suppressed branching programs is equal to the class of decision problems solvable by a nonuniform family of polynomial-size read-once (deterministic) branching programs. When we prove the third result, we also give some insight of the reason why the class of decision problems solvable by a nonuniform family of polynomial-size zero-suppressed branching programs may not be equal to $\mathsf{L/poly}$ (Sect. 3.3).

In the second part of this paper, we attempt to generalize the concept of zero-suppression. We call the basic idea of ZDDs *zero-suppression*. We expect that zero-suppression is effective in computing a certain kind of Boolean functions such as almost all outputs are 0. Zero-suppressed branching programs, which we defined in the first part, are the zero-suppressed version of branching programs. We show that zero-suppression can be applied to other two classical computation models, decision trees and Boolean formulas.

The first computation model is decision trees. We consider zero-suppression for this model. For randomized computation and quantum computation, variants of decision trees (i.e., randomized decision trees and quantum decision trees, respectively) have been well-studied. We define zero-suppressed decision trees and show some gaps of the complexity to deterministic decision trees. Although our results for this model are quite simple observations, it implies a difference between zero-suppression and other computations, and motivates the study of zero-suppression.

The second computation model is Boolean formulas. For decision trees and branching programs, we can smoothly define the zero-suppressed versions, since

the definitions of them are close to ZDDs. We apply zero-suppression to operations of Boolean formulas, and define a new operation. We show an example that Boolean formulas with zero-suppression are effective. Although our results for this model are also quite simple observations, it implies that the concept of zero-suppression could be widely applied.

2 Preliminaries

A Boolean function is a function $f : \{0,1\}^n \to \{0,1\}$.

A *(deterministic) branching program* or *binary decision diagram (BDD)* is a directed acyclic graph. The nodes of out-degree 2 are called *inner nodes* and labeled by a variable. The nodes of out-degree 0 are called *sinks* and labeled by 0 or 1. For each inner node, one of the outgoing edges is labeled by 0 and the other one is labeled by 1. There is a single specific node called the *start node*. An assignment to the variables determines a computation path from the start node to a sink node. The value of the sink node is the output of the branching program or BDD.

A *zero-suppressed binary decision diagram (ZDD)* is also a directed acyclic graph defined in the same way as BDD, and the only difference is the way to determine the output. An assignment to the variables determines a computation path from the start node to a sink node. The ZDD outputs 1 iff the value of the sink node is 1 and all variables which are not contained in the computation path are assigned by 0.

Notice that we define BDD and ZDD with no restriction to the appearance of the variables. (BDD and ZDD often mean the ordered one, i.e., the variable order is fixed and each variable appears at most once on each path.) The *size* of branching programs is the number of its nodes. If the nodes are arranged into a sequence of levels with edges going only from one level to the next, then the *width* is the size of the largest level. A branching program is called *read-once* branching program if each path contains at most one node labeled by each variable.

Decision trees can be defined along the definition of branching programs. We use this way in this paper. A *(deterministic) decision tree* is a branching program whose graph is a rooted tree. The start node of a decision tree is the root. We define the *(deterministic) decision tree complexity* of f, denoted by $D(f)$, as the depth of an optimal (i.e., minimal-depth) decision tree that computes f.

For a nonnegative integer i, NC^i is the class of decision problems solvable by a uniform family of Boolean circuits with polynomial size, depth $O(\log^i n)$, and fan-in 2.

3 Zero-Suppressed Branching Programs

Branching programs and BDDs have a same definition as we defined in Sect. 2. We define *zero-suppressed branching programs* as it has the same definition to ZDDs. In this section, we consider the computational power of zero-suppressed branching programs.

3.1 Constant-Width Zero-Suppressed Branching Programs

Firstly, we prove two lemmas, which are used also in the following subsection.

Lemma 1. *Any deterministic branching program of n variables, size s, and width w can be converted to a zero-suppressed branching programs of size $s + n$ and width w.*

Proof. Let G be a deterministic branching program of n variables, size s, and width w. We convert G to a zero-suppressed branching program as follows. We add n nodes, $v_1, v_2, \ldots, v_n$, such that v_i is labeled by x_i for $1 \leq i \leq n$, and connect two outgoing edges of v_i to v_{i+1} for $1 \leq i \leq n-1$, and connect two outgoing edges of v_n to the 1-sink, and connect all edges which are connected to the 1-sink to v_1.

In the obtained zero-suppressed branching program, every computation path to the 1-sink contains all variables. Thus, by the definition of zero-suppressed branching programs, G and the obtained zero-suppressed branching program compute the same Boolean function. □

Lemma 2. *Any zero-suppressed branching programs of n variables, polynomial size, and width w can be converted to a Boolean circuit of polynomial size and depth $O(\log w \log n)$.*

Proof. We extend the proof of one direction of the Barrington's theorem. A deterministic branching program of n variables, polynomial size, and width 5 can be converted to a Boolean circuit of polynomial size and depth $O(\log n)$ as follows. Two levels of a deterministic branching program are composed to one level by a circuit of a constant depth. Doing this in parallel and repeating it $O(\log n)$ times yield the desired circuit of depth $O(\log n)$.

If the width is w, two levels of a deterministic branching program are composed to one level by a circuit of depth $O(\log w)$. For the case of zero-suppressed branching programs, we need to memorize the variables contained in the computation path, which can be done with no increase of the depth of the circuit.

Actual encoding of each level is as follows. At most w nodes of each level can be numbered with $\lceil \log w \rceil$ bits. For each outgoing edge of each node of a level, $\lceil \log w \rceil + n$ bits are assigned. The first $\lceil \log w \rceil$ bits represent the node which the outgoing edge connects to. The other n bits represent whether each of n variables is contained in the computation path when the outgoing edge is used in computation. □

For the case that the width of zero-suppressed branching programs is a constant, we determine that the equivalent class is NC^1, which is an analog of the Barrington's theorem [1] for deterministic branching programs.

Theorem 1. *For any constant $w \geq 5$, the class of decision problems solvable by a nonuniform family of polynomial-size width w zero-suppressed branching programs is equal to nonuniform NC^1.*

Proof. All problems in nonuniform NC^1 can be solvable by a nonuniform family of polynomial-size width 5 deterministic branching programs [1]. By Lemma 1, the problems can be solvable also by a nonuniform family of polynomial-size width 5 zero-suppressed branching programs. Thus, the class contains nonuniform NC^1.

Consider a problem solvable by a nonuniform family of polynomial-size width w zero-suppressed branching programs. By Lemma 2, the problem is also solvable by a nonuniform family of Boolean circuits of polynomial size and depth $O(\log w \log n)$. Since w is a constant, the class is contained in nonuniform NC^1. □

3.2 General Zero-Suppressed Branching Programs

The main question for zero-suppressed branching programs is whether the class of decision problems solvable by a nonuniform family of polynomial-size zero-suppressed branching programs is equal to $\mathsf{L/poly}$ or not. We show a weaker result.

Theorem 2. *The class of decision problems solvable by a nonuniform family of polynomial-size zero-suppressed branching programs contains* $\mathsf{L/poly}$*, and is contained in nonuniform* NC^2*.*

Proof. All problems in $\mathsf{L/poly}$ can be solvable by a nonuniform family of polynomial-size deterministic branching programs [3]. By Lemma 1, the problems can be solvable also by a nonuniform family of polynomial-size zero-suppressed branching programs. Thus, the class contains $\mathsf{L/poly}$.

Consider a problem solvable by a nonuniform family of polynomial-size zero-suppressed branching programs. Obviously, the width of the zero-suppressed branching programs is a polynomial of n. Thus, by Lemma 2, the problem is also solvable by a nonuniform family of Boolean circuits of polynomial size and depth $O(\log^2 n)$. Therefore, the class is contained in nonuniform NC^2. □

3.3 Read-Once Zero-Suppressed Branching Programs

In deterministic branching programs, the states in computation are decided only by the node which was reached in computation. Thus, the number of the states is at most the size of the branching program, and, if the size is at most polynomial, then each state can be represented by logarithmic space, which leads to the fact that the class of decision problems solvable by a nonuniform family of polynomial-size deterministic branching programs is equal to $\mathsf{L/poly}$. On the other hand, in zero-suppressed branching programs, the states in computation are not decided only by the node which was reached in computation. It depends on the variables which were contained in the computation path. This is the main reason why the class of decision problems solvable by a nonuniform family of polynomial-size zero-suppressed branching programs may not be equal to $\mathsf{L/poly}$. Note that the information of the passed variables cannot be saved in logarithmic space.

In this subsection, we consider a simple case. If deterministic and zero-suppressed branching programs are read-once, then we can convert them to each other with polynomial increase of the size.

Theorem 3. *The class of decision problems solvable by a nonuniform family of polynomial-size read-once zero-suppressed branching programs is equal to the class of decision problems solvable by a nonuniform family of polynomial-size read-once deterministic branching programs.*

Proof. We prove two lemmas.

Lemma 3. *Any read-once deterministic branching program of n variables and size s can be converted to a read-once zero-suppressed branching program of size $s + 2ns$.*

Proof. Note that the way of the proof of Lemma 1 does not give a read-once zero-suppressed branching program. We need more consideration to the place where new nodes are added.

Let G be a read-once deterministic branching program of n variables and size s. Let $v_1, v_2, \ldots, v_s$ be the nodes in G such that $v_1, v_2, \ldots, v_s$ is a topologically sorted order. We convert G so that every computation path which reaches to a node contains the same all variables, for each node from v_1 to v_s. Assume that every computation path which reaches to v_i contains the same variables for each $1 \leq i \leq k-1$. We convert G so that every computation path which reaches to v_k contains the same variables as follows. Let X_i be the set of variables which are contained in computation paths to v_i, for $1 \leq i \leq k-1$. Let X be the union of X_j such that there is an edge from v_j to v_k. Let $X'_i = X - X_i$. For every edge e from v_i to v_k, $1 \leq i \leq k-1$, we add $|X'_i|$ nodes, $u_1, u_2, \ldots, u_{|X'_i|}$, such that the nodes are labeled by the variables contained in X'_i, and connect two outgoing edges of u_j to u_{j+1} for $1 \leq j \leq |X'_i| - 1$, and connect two outgoing edges of $u_{|X'_i|}$ to v_k, and connect e to u_1. Let G' be the obtained branching program. If computation paths to the 1-sink in G' do not contain all variables, we modify G' to contain all variables by a similar way.

G' is read-once, since added nodes are labeled by the variables contained in X'_i. In G', every computation path to the 1-sink contains all variables. Thus, by the definition of zero-suppressed branching programs, G and G' compute the same Boolean function. The number of added node is at most n for each edge. □

Lemma 4. *Any read-once zero-suppressed branching program of n variables and size s can be converted to a read-once deterministic branching program of size $s + 2ns$.*

Proof. Let G be a read-once zero-suppressed branching program of n variables and size s. Let $v_1, v_2, \ldots, v_s$ be the nodes in G such that $v_1, v_2, \ldots, v_s$ is a topologically sorted order. We convert G so that every computation path which reaches to a node contains the same all variables, for each node from v_1 to v_s. Assume that every computation path which reaches to v_i contains the same variables for each $1 \leq i \leq k-1$. We convert G so that every computation path

which reaches to v_k contains the same variables as follows. Let X_i be the set of variables which are contained in computation paths to v_i, for $1 \leq i \leq k-1$. Let X be the union of X_j such that there is an edge from v_j to v_k. Let $X'_i = X - X_i$. For every edge e from v_i to v_k, $1 \leq i \leq k-1$, we add $|X'_i|$ nodes, $u_1, u_2, \ldots, u_{|X'_i|}$, such that the nodes are labeled by the variables contained in X'_i, and connect the outgoing 0-edge of u_j to u_{j+1} for $1 \leq j \leq |X'_i| - 1$, and connect the outgoing 0-edge of $u_{|X'_i|}$ to v_k, and connect the outgoing 1-edge of u_j to the 0-sink for $1 \leq j \leq |X'_i|$, and connect e to u_1. Let G' be the obtained branching program. If computation paths to the 1-sink in G' do not contain all variables, we modify G' to contain all variables by a similar way.

G' is read-once, since added nodes are labeled by the variables contained in X'_i. By the definition of zero-suppressed branching programs, G and G' compute the same Boolean function. The number of added node is at most n for each edge. □

By the two lemmas, the theorem holds. □

4 Zero-Suppression and Other Computation Models

In this section, we attempt to generalize the concept of zero-suppression. We show that zero-suppression can be applied to other two classical computation models, decision trees and Boolean formulas.

4.1 Zero-Suppressed Decision Trees

Since a decision tree is a branching program whose graph is a rooted tree, zero-suppressed decision trees are naturally defined as follows.

A *zero-suppressed decision tree* is also a rooted tree defined in the same way as deterministic decision tree, and the only difference is the way to determine the output. An assignment to the variables determines a computation path from the start node to a sink node. The zero-suppressed decision tree outputs 1 iff the value of the sink node is 1 and all variables which are not contained in the computation path are assigned by 0. We define the *zero-suppressed decision tree complexity* of f, denoted by $Z(f)$, as the depth of an optimal (i.e., minimal-depth) zero-suppressed decision tree that computes f.

We can immediately obtain the following gaps.

Theorem 4. *There is a Boolean function f such that $D(f) = 0$ and $Z(f) = n$.*

Proof. Let $f = 1$. (See also Fig. 1.) □

Theorem 5. *There is a Boolean function g such that $D(g) = n$ and $Z(g) = 0$.*

Proof. Let $g = \neg x_1 \wedge \neg x_2 \wedge \cdots \wedge \neg x_n$. (See also Fig. 2.) □

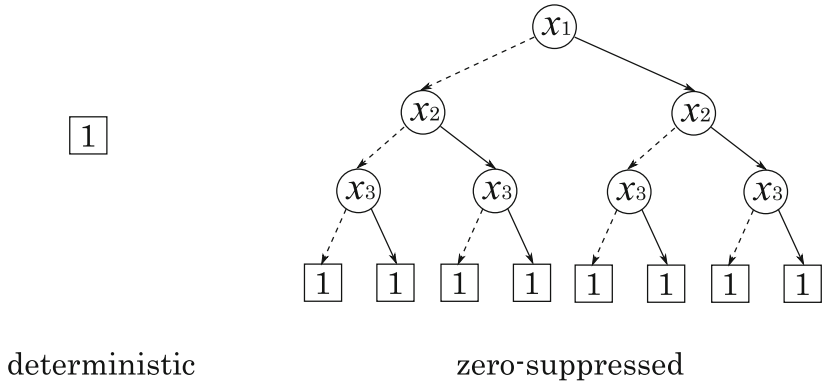


Fig. 1. Decision trees computing f for $n = 3$

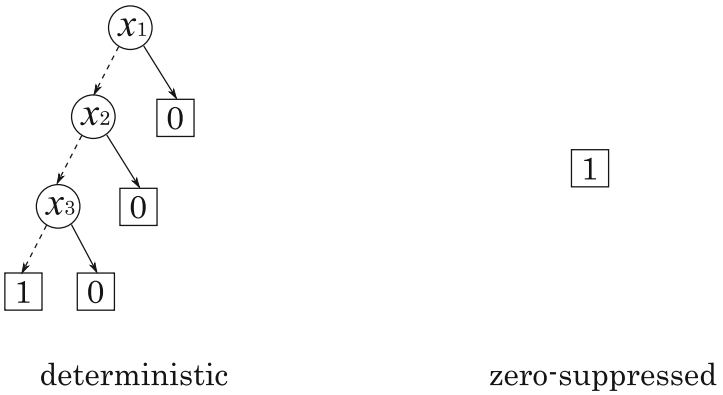


Fig. 2. Decision trees computing g for $n = 3$

Thus, the advantages and disadvantages of deterministic and zero-suppressed decision trees strongly depend on the Boolean function which decision trees compute. Although these two theorems are quite simple observations, the difference from other computations implies unique behavior of zero-suppression. See the following proposition.

Proposition 1. $Q_2(f) \leq R_2(f) \leq D(f)$.

$Q_2(f)$ and $R_2(f)$ are variants of decision tree complexity with quantum computation and randomized computation, respectively. For the definitions and the more details, we refer to Sect. 3 of the survey paper [2].

4.2 Boolean Formulas with Zero-Suppression

In this subsection, we do not intend to give formal definitions. We show a simple idea for future works.

The exactly-k-function $E_k^n(x_1, \ldots, x_n)$ is 1 iff $\Sigma_{i=1}^n x_i = k$. In the standard formulas, an obvious representation of E_1^3 is

$$(x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge \neg x_2 \wedge x_3).$$

In formulas with zero-suppression, E_1^3 is simply represented by

$$(x_1)^z \vee (x_2)^z \vee (x_3)^z,$$

where $()^z$ is a new operation which we define from the concept of zero-suppression and $(f)^z$ is 1 iff $f = 1$ and all variables which are not contained in f are assigned by 0.

Although Boolean formulas are a computation model to compute Boolean functions in computational complexity theory, it also appears in various areas. From the example of Boolean formulas, we note that zero-suppression is not necessarily considered only for computational complexity theory, although it is beyond the main aim of this paper.

5 Conclusions

In this paper, we investigated zero-suppression. We applied zero-suppression to three computation models including branching programs. It raises a question: More generally, can we establish zero-suppressed computation as a new computation? We hope that this paper will be the first step to study zero-suppressed computation. On the other hand, we also noted that zero-suppression is not necessarily considered only for computational complexity theory with an example of Boolean formulas.

While three computation models in this paper need further studies, a challenging open problem is to seek another computation model whose zero-suppressed version is meaningful, hopefully also in the real world. When we consider other computation models (e.g., Boolean circuits), it is a difficult and interesting problem even to define the appropriate zero-suppressed version.

For zero-suppressed branching programs, it remains open whether the class of decision problems solvable by a nonuniform family of polynomial-size zero-suppressed branching programs is equal to $\mathsf{L/poly}$ or not. We showed some related results to the question in this paper. By Theorem 2, there are the following four cases.

- The class is equal to $\mathsf{L/poly}$.
- The class is equal to nonuniform NC^2.
- The class is equal to another known complexity class between $\mathsf{L/poly}$ and nonuniform NC^2.
- The class is not equal to any known complexity class.

Our observation for zero-suppressed decision trees implies unusual properties of zero-suppression, which makes us feel the possibility of some new complexity class.

References

1. Barrington, D.A.M.: Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. J. Comput. Syst. Sci. **38**(1), 150–164 (1989)
2. Buhrman, H., de Wolf, R.: Complexity measures and decision tree complexity: a survey. Theor. Comput. Sci. **288**(1), 21–43 (2002)
3. Cobham, A.: The recognition problem for the set of perfect squares. In: Proceedings of the 7th Annual Symposium on Switching and Automata Theory, pp. 78–87 (1966)
4. Knuth, D.E.: The Art of Computer Programming, Volume 4, Fascicle 1. Addison-Wesley, Boston (2009)
5. Meinel, C., Theobald, T.: Algorithms and Data Structures in VLSI Design: OBDD - Foundations and Applications. Springer, Heidelberg (1998). https://doi.org/10.1007/978-3-642-58940-9
6. Minato, S.: Zero-suppressed BDDs for set manipulation in combinatorial problems. In: Proceedings of DAC, pp. 272–277 (1993)
7. Wegener, I.: Branching Programs and Binary Decision Diagrams. SIAM, Philadelphia (2000)

The Crossing Number of Seq-Shellable Drawings of Complete Graphs

Petra Mutzel and Lutz Oettershagen(✉)

Department of Computer Science, TU Dortmund University, Dortmund, Germany
{petra.mutzel,lutz.oettershagen}@tu-dortmund.de

Abstract. The Harary-Hill conjecture states that for every $n \geq 3$ the number of crossings of a drawing of the complete graph K_n is at least

$$H(n) := \frac{1}{4}\left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{n-1}{2} \right\rfloor \left\lfloor \frac{n-2}{2} \right\rfloor \left\lfloor \frac{n-3}{2} \right\rfloor.$$

So far, the conjecture could only be verified for arbitrary drawings of K_n with $n \leq 12$. In recent years, progress has been made in verifying the conjecture for certain classes of drawings, for example 2-page-book, x-monotone, x-bounded, shellable and bishellable drawings. Up to now, the class of bishellable drawings was the broadest class for which the Harary-Hill conjecture has been verified, as it contains all beforehand mentioned classes. In this work, we introduce the class of *seq-shellable* drawings and verify the Harary-Hill conjecture for this new class. We show that bishellability implies seq-shellability and exhibit a non-bishellable but seq-shellable drawing of K_{11}, therefore the class of seq-shellable drawings strictly contains the class of bishellable drawings.

1 Introduction

Let $G = (V, E)$ be an undirected graph and K_n the complete graph on $n > 0$ vertices. The crossing number $cr(G)$ of G is the smallest number of edge crossings over all possible drawings of G. In a drawing D every vertex $v \in V$ is represented by a point and every edge $uv \in E$ with $u, v \in V$ is represented by a simple curve connecting the corresponding points of u and v. The Harary-Hill conjecture states the following.

Conjecture 1 (Harary-Hill [1]). Let K_n be the complete graph with n vertices, then

$$cr(K_n) = H(n) := \frac{1}{4}\left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{n-1}{2} \right\rfloor \left\lfloor \frac{n-2}{2} \right\rfloor \left\lfloor \frac{n-3}{2} \right\rfloor.$$

There are construction methods for drawings of K_n that lead to exactly $H(n)$ crossings, for example the class of *cylindrical* drawings first described by Harary and Hill [2]. For a cylindrical drawing, we put $\lfloor \frac{n}{2} \rfloor$ vertices on the top rim and the remaining $\lceil \frac{n}{2} \rceil$ vertices on the bottom rim of a cylinder. Edges between vertices on the same rim (lid or bottom) are connected with straight lines on

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 273–284, 2018.
https://doi.org/10.1007/978-3-319-94667-2_23

the lid or bottom. Two vertices on opposite rims are connected with an edge along the geodesic between the two vertices. The drawing of K_6 in Fig. 1(a) is homeomorphic to a planarized cylindrical drawing of K_6.

However, there is no proof for the lower bound of the conjecture for arbitrary drawings of K_n with $n > 12$. The cases for $n \leq 10$ are shown by Guy [1] and for $n = 11$ by Pan and Richter [3]. Guy [1] argues that $cr(K_{2n+1}) \geq H(2n+1)$ implies $cr(K_{2(n+1)}) \geq H(2(n+1))$, hence $cr(K_{12}) \geq H(12)$. McQuillan et al. showed that $cr(K_{13}) \geq 219$ [4]. Ábrego et al. [5] improved the result to $cr(K_{13}) \in \{223, 225\}$.

Beside these results for arbitrary drawings, there has been success in proving the Harary-Hill conjecture for different classes of drawings. So far, the conjecture has been verified for 2-page-book [6], x-monotone [7–9], x-bounded [7], shellable [7] and bishellable drawings [10]. The class of bishellable drawings comprises all beforehand mentioned classes, and until now it was the largest class of drawings for which the Harary-Hill conjecture has been verified. Ábrego et al. [10] showed that the Harary-Hill conjecture holds for bishellable drawings using cumulated k-edges.

Our Contribution. In this work, we introduce the new class of *seq-shellable* drawings and verify the Harary-Hill conjecture for this new class. We show that bishellability implies seq-shellability and exhibit a drawing of K_{11} which is seq-shellable but not bishellable. Therefore, we establish that the class of seq-shellable drawings strictly contains the class of bishellable drawings.

The outline of this paper is as follows. In Sect. 2 we present the preliminaries, and in particular the background on k-edges, cumulated k-edges and their usage for verifying the Harary-Hill conjecture. In Sect. 3 we define *simple sequences* and their usage for proving lower bounds on the number of invariant edges. We present the definition of seq-shellability, verify the Harary-Hill conjecture for the new class and show its superiority towards the class of bishellable drawings. Finally, in Sect. 4 we draw our conclusion and close with open questions.

2 Preliminaries

Formally, a *drawing* D of a graph G on the plane is an injection ϕ from the vertex set V into the plane, and a mapping of the edge set E into the set of simple curves, such that the curve corresponding to the edge $e = uv$ has endpoints $\phi(u)$ and $\phi(v)$, and contains no other vertices [11]. We call an intersection point of the interior of two edges a crossing and a shared endpoint of two adjacent edges is not considered a crossing. The crossing number $cr(D)$ of a drawing D equals the number of crossings in D and the crossing number $cr(G)$ of a graph G is the minimum crossing number over all its possible drawings. We restrict our discussions to *good* drawings of K_n, and call a drawing *good* if (1) any two of the curves have finitely many points in common, (2) no two curves have a point in common in a tangential way, (3) no three curves cross each other in the same point, (4) any two edges cross at most once and (5) no two adjacent edges cross.

It is known that every drawing with a minimum number of crossings is good [12]. In the discussion of a drawing D, we call the points also vertices, the curves edges and V denotes the set of vertices (i.e. points), and E denotes the edges (i.e. curves) of D. If we subtract the drawing D from the plane, a set of open discs remain. We call $\mathcal{F}(D) := \mathbb{R}^2 \setminus D$ the set of *faces* of the drawing D. If we remove a vertex v and all its incident edges from D, we get the subdrawing $D-v$. Moreover, we might consider the drawing to be on the surface of the sphere S^2, which is equivalent to the drawing on the plane due to the homeomorphism between the plane and the sphere minus one point.

In [10] Ábrego et al. introduce bishellable drawings.

Definition 1 (Bishellability [10]). *For a non-negative integer s, a drawing D of K_n is s-bishellable if there exist sequences $a_0, a_1, \ldots, a_s$ and $b_s, b_{s-1}, \ldots, b_1, b_0$, each sequence consisting of distinct vertices of K_n, so that with respect to a reference face F:*

(i) For each $i \in \{0, \ldots, s\}$, the vertex a_i is incident to the face of $D - \{a_0, a_1, \ldots, a_{i-1}\}$ that contains F,
(ii) for each $i \in \{0, \ldots, s\}$, the vertex b_i is incident to the face of $D - \{b_0, b_1, \ldots, b_{i-1}\}$ that contains F, and
(iii) for each $i \in \{0, \ldots, s\}$, the set $\{a_0, a_1, \ldots a_i\} \cap \{b_{s-i}, b_{s-i-1}, \ldots, b_0\}$ is empty.

The class of bishellable drawings contains all drawings that are $(\lfloor \frac{n}{2} \rfloor - 2)$-bishellable. In order to show that if a drawing D is $(\lfloor \frac{n}{2} \rfloor - 2)$-bishellable, the Harary-Hill conjecture holds for D, Ábrego et al. use the notion of k-edges. The origins of k-edges lie in computational geometry and problems over n-point set, especially problems on halving lines and k-set [13]. An early definition in the geometric setting goes back to Erdős et al. [14]. Given a set P of n points in general position in the plane, the authors add a directed edge $e = (p_i, p_j)$ between the two distinct points p_i and p_j, and consider the continuation as line that separates the plane into the left and right half plane. There is a (possibly empty) point set $P_L \subseteq P$ on the left side of e, i.e. left half plane. Erdős et al. assign $k := \min(|P_L|, |P \setminus P_L|)$ to e. Later, the name k-edge emerged and Lovász et al. [15] used k-edges for determining a lower bound on the crossing number of rectilinear graph drawings. Finally, Ábrego et al. [6] extended the concept of k-edges from rectilinear to topological graph drawings.

Every edge in a good drawing D of K_n is a k-edge with $k \in \{0, \ldots, \lfloor \frac{n}{2} \rfloor - 1\}$. Let D be on the surface of the sphere S^2, and $e = uv$ be an edge in D and $F \in \mathcal{F}(D)$ be an arbitrary but fixed face; we call F the *reference face*. Together with any vertex $w \in V \setminus \{u, v\}$, the edge e forms a triangle uvw and hence a closed curve that separates the surface of the sphere into two parts. For an arbitrary but fixed orientation of e one can distinguish between the left part and the right part of the separated surface. If F lies in the left part of the surface, we say the triangle has orientation $+$ else it has orientation $-$. For e there are $n-2$ possible triangles in total, of which $0 \leq i \leq n-2$ triangles have orientation $+$ (or $-$) and $n-2-i$ triangles have orientation $-$ (or $+$ respectively). We define

$k := \min(i, n-2-i)$ and say e is an *k-edge* with respect to the reference face F and its *k-value* equals k with respect to F. Ábrego et al. [6] show that the crossing number of a drawing is expressible in terms of the number of k-edges for $0 \leq k \leq \lfloor \frac{n}{2} \rfloor - 1$ with respect to the reference face. The following definition of the *cumulated* number of k-edges is helpful in determining the lower bound of the crossing number.

Definition 2 (Cumulated k-edges [6]). *Let D be good drawing and $E_k(D)$ be the number of k-edges in D with respect to a reference face $F \in \mathcal{F}(D)$ and for $k \in \{0, \ldots, \lfloor \frac{n}{2} \rfloor - 1\}$. We call*

$$E_{\leq\leq k}(D) := \sum_{i=0}^{k} (k+1-i) E_i(D)$$

the cumulated number of k-edges with respect to F.

We also write *cumulated k-edges* or *cumulated k-value* instead of cumulated number of k-edges. Lower bounds on $E_{\leq\leq k}(D)$ for $0 \leq k \leq \lfloor \frac{n}{2} \rfloor - 2$ translate directly into a lower bound for $cr(D)$.

Lemma 1 [6]**.** *Let D be a good drawing of K_n and $F \in \mathcal{F}(D)$. If $E_{\leq\leq k}(D) \geq 3\binom{k+3}{3}$ for all $0 \leq k \leq \lfloor \frac{n}{2} \rfloor - 2$ with respect to F, then $cr(D) \geq H(n)$.* □

If a vertex v is incident to the reference face, the edges incident to v have a predetermined distribution of k-values.

Lemma 2 [6]**.** *Let D be a good drawing of K_n, $F \in \mathcal{F}(D)$ and $v \in V$ be a vertex incident to F. With respect to F, vertex v is incident to two i-edges for $0 \leq i \leq \lfloor \frac{n}{2} \rfloor - 2$. Furthermore, if we label the edges incident to v counter clockwise with $e_0, \ldots, e_{n-2}$ such that e_0 and e_{n-2} are incident to the face F, then e_i is a k-edge with $k = \min(i, n-2-i)$ for $0 \leq i \leq n-2$.* □

Examples for Lemma 2 are the vertices incident to F in Fig. 1. We denote the cumulated k-values for edges incident to a vertex v in a drawing D with $E_{\leq\leq k}(D, v)$. Due to Lemma 2 it follows that $E_{\leq\leq k}(D, v) = \sum_{i=0}^{k} (k+1-i) \cdot 2 = 2\binom{k+2}{2}$.

Next, we introduce *invariant* k-edges. Consider removing a vertex $v \in V$ from a good drawing D of K_n, resulting in the subdrawing $D - v$. By deleting v and its incident edges every remaining edge loses one triangle, i.e. for an edge $uw \in E$ there are only $(n-3)$ triangles uwx with $x \in V \setminus \{u, v\}$ (instead of the $(n-2)$ triangles in drawing D). The k-value of any edge $e \in E$ is defined as the minimum count of $+$ or $-$ oriented triangles that contain e. If the lost triangle had the same orientation as the minority of triangles, the k-value of e is reduced by one else it stays the same. Therefore, every k-edge in D with respect to $F \in \mathcal{F}(D)$ is either a k-edge or a $(k-1)$-edge in the subdrawing $D - v$ with respect to $F' \in \mathcal{F}(D - v)$ and $F \subseteq F'$. We call an edge e *invariant* if e has the same k-value with respect to F in D as for F' in D'. We denote the number of cumulated invariant k-edges between D and D' (with respect to F and F'

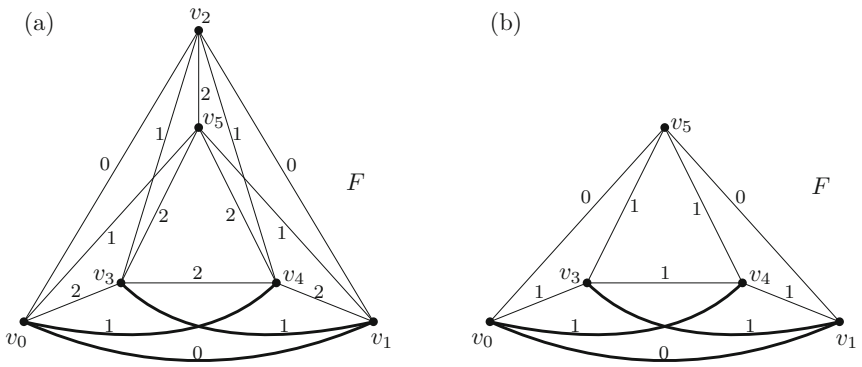


Fig. 1. Example: (a) shows a crossing optimal drawing D of K_6 with the k-values at the edges. (b) shows the subdrawing $D - v_2$ and its k-values. The fat highlighted edges v_0v_1, v_0v_4 and v_1v_3 are invariant and keep their k-values. The reference face is the outer face F.

respectively) with $I_{\leq k}(D, D')$, i.e. $I_{\leq k}(D, D')$ equals the sum of the number of invariant i-edges for $0 \leq i \leq k$.

For a good drawing D of K_n, we are able to express the value of cumulated k-edges with respect to a reference face $F \in \mathcal{F}(D)$ recursively by adding up the cumulated $(k-1)$-value of a subdrawing $D - v$, the contribution of the edges incident to v and the number of *invariant edges* between D and $D - v$.

Lemma 3 [10]**.** *Let D be a good drawing of K_n, $v \in V$ and $F \in \mathcal{F}(D)$. With respect to the reference face F, we have*

$$E_{\leq\leq k}(D) = E_{\leq\leq k-1}(D - v) + E_{\leq\leq k}(D, v) + I_{\leq k}(D, D - v).$$

□

Ábrego et al. [10] use an inductive proof over k to show that for a bishellable drawing D of K_n $E_{\leq\leq k}(D) \geq 3\binom{k+3}{3}$ for all $k \in \{0, \ldots, \lfloor \frac{n}{2} \rfloor - 2\}$. Together with Lemma 1 follows $cr(D) \geq H(n)$.

Here, we also use Lemma 3 and show that for a seq-shellable drawing D of K_n the lower bounds on $E_{\leq\leq k}(D)$ hold for all $k \in \{0, \ldots, \lfloor \frac{n}{2} \rfloor - 2\}$. But in contrast to [10], we use a more general and at the same time easy to follow approach to guarantee lower bounds on the number of invariant edges $I_{\leq k}(D, D - v)$ for $0 \leq k \leq \lfloor \frac{n}{2} \rfloor - 2$.

3 Seq-Shellability

Before we proceed with the definition of seq-shellability, we introduce simple sequences.

3.1 Simple Sequences

We use simple sequences to guarantee a lower bound of the number of invariant edges in the recursive formulation of the cumulated k-value.

Definition 3 (Simple sequence). *Let D be a good drawing of K_n, $F \in \mathcal{F}(D)$ and $v \in V$ with v incident to F. Furthermore, let $S_v = (u_0, \ldots, u_k)$ with $u_i \in V \setminus \{v\}$ be a sequence of distinct vertices. If u_0 is incident to F and vertex u_i is incident to a face containing F in subdrawing $D - \{u_0, \ldots, u_{i-1}\}$ for all $1 \leq i \leq k$, then we call S_v simple sequence of v.*

Before we continue with a result for lower bounds on the number of invariant edges using simple sequences, we need the following lemma.

Lemma 4. *Let D be a good drawing of K_n, $F \in \mathcal{F}(D)$ and $u, v \in V$ with u and v incident to F. The edge uv touches F either over its full length or not at all (except its endpoints).*

Proof. Assume that D a is good drawing of K_n in which the edge uv touches F only partly. We can exclude the case that an edge cuts a part out of uv by crossing it more than once due to the goodness of the drawing (see Fig. 2(a)). The case that an edge crosses the whole face F and separates it into two faces is also impossible, because this would contradict that both u and v are incident to F. Therefore, a vertex x has to be on the same side of uv as F and a vertex y on the other side such that the edge xy crosses uv. But the edge xu cannot cross any edge uz with $z \in V \setminus \{u\}$ as this would contradict the goodness of D and xu cannot leave the superface of x without separating v from F (see Fig. 2(b) and (c)). We have the symmetric case for v. Consequently, uv cannot touch F beside its endpoints u and v (see Fig. 2(d)), a contradiction to the assumption. □

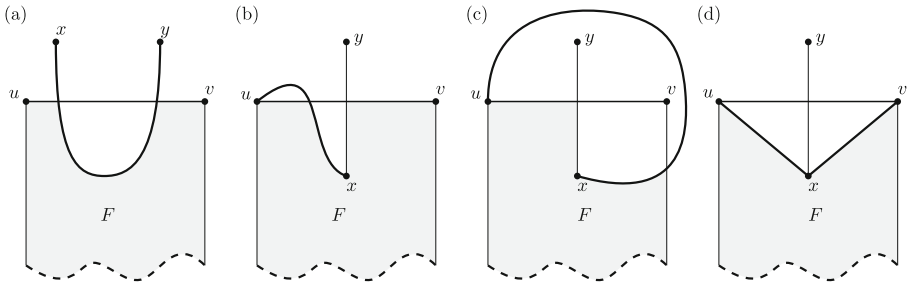


Fig. 2. (a) Due to the goodness of D an edge cannot *cut* a part out of the edge uv. (b) The edges uv and ux cross, both have vertex u as endpoint thus the drawing is not good. (c) The drawing is good but vertex v is not incident to the face F. (d) The edge uv is crossed, the drawing is good and both vertices u and v is are incident to F, however uv is not incident to F.

Corollary 1. *Let D be a good drawing of K_n, $F \in \mathcal{F}(D)$ and $u, v \in V$ with both u and v incident to F. If and only if uv is a j-edge, there are exactly j or $n-2-j$ vertices on the same side of uv as the reference face F.* □

The following lemma provides a lower bound for the number of invariant edges in the case that F is incident to at least two vertices and we remove one of them.

Lemma 5. *Let D be a good drawing of K_n, $F \in \mathcal{F}(D)$ and $v, w \in V$ with v and w incident to F. If we remove v from D, then w is incident to at least $\lfloor \frac{n}{2} \rfloor - 1$ invariant edges.*

Proof. We label the edges incident to w counter clockwise with $e_0, \ldots, e_{n-2}$ such that e_0 and e_{n-2} are incident to the face F, and we label the vertex at the other end of e_i with u_i. Furthermore, we orient all edges incident to w as outgoing edges. Due to Lemma 2 we know that w has two i-edges for $0 \leq i \leq \lfloor \frac{n}{2} \rfloor - 2$. Edge e_i obtains its i-value from the minimum of say $+$ oriented triangles and edge e_{n-2-i} obtains its i-value from the minimum $-$ oriented triangles (or vice versa). Assume that vw is incident to F, i.e. vw is a 0-edge and all triangles vwu for $u \in V \setminus \{v, w\}$ have the same orientation. Consequently, all e_i or all e_{n-2-i} for $0 \leq i \leq \lfloor \frac{n}{2} \rfloor - 2$ are invariant. In the case that vw is not incident to F and is a j-edge, there are j triangles vwu_h with $u_h \in V \setminus \{v, w\}$, $0 \leq h \leq j-1$ or $n-1-j \leq h \leq n-2$ and u_h is on the same side of vw as F (Corollary 1). This means, each triangle wu_hv is part of the majority of orientations for the k-value of edge wu_h, therefore removing v does not change the k-value and there are j additional invariant edges incident to w if we remove v. □

The following lemma provides a lower bound for the number of cumulated invariant k-edges if we remove a vertex that has a simple sequence.

Lemma 6. *Let D be a good drawing of K_n, $F \in \mathcal{F}(D)$ and $v \in V$ with v incident to F. If v has a simple sequence $S_v = (u_0, \ldots, u_k)$, then*

$$I_{\leq k}(D, D - v) \geq \binom{k+2}{2}$$

with respect to F and for all $k \in \{0, \ldots, \lfloor \frac{n}{2} \rfloor - 2\}$.

Proof. Let $k \in \{0, \ldots, \lfloor \frac{n}{2} \rfloor - 2\}$. We know that u_0 has at least $k+1 \leq \lfloor \frac{n}{2} \rfloor - 1$ invariant edges with respect to F and removing v. After removing vertex u_0 from drawing D, vertices v and u_1 are incident to F. Since $k \leq \lfloor \frac{n}{2} \rfloor - 2 \leq \lfloor \frac{n-1}{2} \rfloor - 1$ and u_0 has an edge to u_1 in drawing D, vertex u_1 has at least k invariant edges with respect to F and removing v in drawing D. In general, after removing vertices $u_0, \ldots, u_{i-1}$ from drawing D, vertices v and u_i are incident to F. For $u \in \{u_0, \ldots, u_{i-1}\}$ the edge uu_i in drawing D may be invariant or non-invariant, and we have $k+1-i \leq \lfloor \frac{n}{2} \rfloor - 1 - i \leq \lfloor \frac{n-i}{2} \rfloor - 1$. Therefore, u_i has at least $k-i+1$

invariant edges in drawing D with respect to F and removing v. Summing up leads to

$$I_{\leq k}(D, D-v) \geq \sum_{i=0}^{k}(k+1-i) = \binom{k+2}{2}. \qquad \square$$

3.2 Seq-Shellable Drawings

With help of simple sequences we define k-seq-shellability. For a sequence of distinct vertices $a_0, \ldots, a_k$ we assign to each vertex a_i with $0 \leq i \leq k \leq n-2$ a simple sequence S_i, under the condition that S_i does not contain any of the vertices $a_0, \ldots, a_{i-1}$.

Definition 4 (Seq-Shellability). *Let D be a good drawing of K_n. We call D k-seq-shellable for $k \geq 0$ if there exists a face $F \in \mathcal{F}(D)$ and a sequence of distinct vertices $a_0, \ldots, a_k$ such that a_0 is incident to F and*

1. *for each $i \in \{1, \ldots, k\}$, vertex a_i is incident to the face containing F in drawing $D - \{a_0, \ldots, a_{i-1}\}$ and*
2. *for each $i \in \{0, \ldots, k\}$, vertex a_i has a simple sequence $S_i = (u_0, \ldots, u_{k-i})$ with $u_j \in V \setminus \{a_0, \ldots, a_i\}$ for $0 \leq j \leq k-i$ in drawing $D - \{a_0, \ldots, a_{i-1}\}$.*

Notice that if D is k-seq-shellable for $k > 0$, then the subdrawing $D - a_0$ is $(k-1)$-seq-shellable. Moreover, if D is k-seq-shellable, then D is also j-seq-shellable for $0 \leq j \leq k$.

Lemma 7. *If D is a good drawing of K_n and D is k-seq-shellable with $k \in \{0, \ldots, \lfloor \frac{n}{2} \rfloor - 2\}$, then $E_{\leq\leq k}(D) \geq 3\binom{k+3}{3}$.*

Proof. We proceed with induction over k. For $k = 0$ the reference face is incident to at least three 0-edges and it follows that

$$E_{\leq\leq 0}(D) \geq 3 = 3\binom{0+3}{3}.$$

For the induction step, let D be k-seq-shellable with $a_0, \ldots, a_k$ and the sequences $S_0, \ldots, S_k$. Consider the drawing $D - a_0$ which is $(k-1)$-seq-shellable for $a_1, \ldots, a_k$ and $S_1, \ldots, S_k$. Since $k - 1 \leq (\lfloor \frac{n}{2} \rfloor - 2) - 1 \leq (\lfloor \frac{n-1}{2} \rfloor - 2)$, we assume

$$E_{\leq\leq k-1}(D - a_0) \geq 3\binom{k+2}{3}.$$

We use the recursive formulation introduced in Lemma 3, i.e.

$$E_{\leq\leq k}(D) = E_{\leq\leq k-1}(D - a_0) + E_{\leq\leq k}(D, a_0) + I_{\leq k}(D, D - a_0).$$

Because a_0 is incident to F, we have $E_{\leq\leq k}(D, a_0) = 2\binom{k+2}{2}$, and with the simple sequence S_0 of a_0 follows $I_{\leq k}(D, D - a_0) \geq \binom{k+2}{2}$ (see Lemma 6). Together with the induction hypothesis, we have

$$E_{\leq\leq k}(D) \geq 3\binom{k+2}{3} + 2\binom{k+2}{2} + \binom{k+2}{2} = 3\binom{k+3}{3}. \qquad \square$$

Using Lemmas 1 and 7, we are able to verify the Harary-Hill conjecture for seq-shellable drawings.

Theorem 1. *If D is a good drawing of K_n and D is $(\lfloor \frac{n}{2} \rfloor - 2)$-seq-shellable, then $cr(D) \geq H(n)$.*

Proof. Let D be a good drawing of K_n and $(\lfloor \frac{n}{2} \rfloor - 2)$-seq-shellable. Since D is $(\lfloor \frac{n}{2} \rfloor - 2)$-seq-shellable, it is also k-seq-shellable for $0 \leq k \leq \lfloor \frac{n}{2} \rfloor - 2$. We apply Lemma 7 and have $E_{\leq \leq k}(D) \geq 3\binom{k+3}{3}$ for $0 \leq k \leq \lfloor \frac{n}{2} \rfloor - 2$ and the result follows with Lemma 1. □

If a drawing D of K_n is $(\lfloor \frac{n}{2} \rfloor - 2)$-seq-shellable, we omit the $(\lfloor \frac{n}{2} \rfloor - 2)$ part and say D is seq-shellable. The class of seq-shellable drawings contains all drawings that are $(\lfloor \frac{n}{2} \rfloor - 2)$-seq-shellable.

Theorem 2. *The class of seq-shellable drawings strictly contains the class of bishellable drawings.*

Proof. First, we show that k-bishellability implies k-seq-shellability. Let D be a k-bishellable drawing of K_n with the associated sequences $a_0, \ldots, a_k$ and $b_0, \ldots, b_k$. In order to show that D is k-seq-shellable, we choose $a_0, \ldots, a_k$ as vertex sequence and k simple sequences S_i for $0 \leq i \leq k$ such that

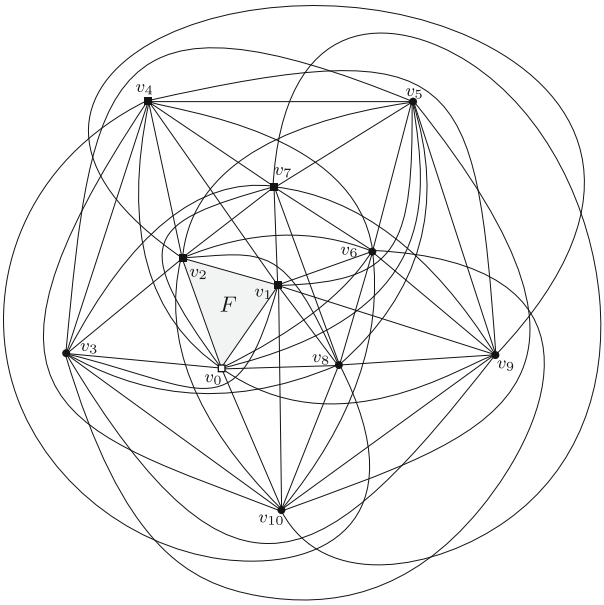


Fig. 3. Drawing H of K_{11} which is not bishellable for any face, however it is seq-shellable for face F, vertex sequence (v_0, v_2, v_3, v_4) and the simple sequences $S_0 = (v_1, v_2, v_7, v_4)$, $S_1 = (v_1, v_8, v_6)$, $S_2 = (v_1, v_8)$ and $S_3 = (v_1)$. Vertex v_0 and the vertices of S_0 are highlighted as unfilled and filled squares.

$S_i = (b_0, \ldots, b_{k-i})$. We assign simple sequence S_i to vertex a_i for each $0 \leq i \leq k$ and see that D is indeed seq-shellable. Furthermore, drawing H of K_{11} in Fig. 3 is not bishellable but seq-shellable. It is impossible to find sequences $a_0, \ldots, a_3$ and $b_0, \ldots, b_3$ in H that fulfill the definition of bishellability. However, H is seq-shellable for face F, vertex sequence (v_0, v_2, v_3, v_4) and the simple sequences $S_0 = (v_1, v_2, v_7, v_4)$, $S_1 = (v_1, v_8, v_6)$, $S_2 = (v_1, v_8)$ and $S_3 = (v_1)$. □

The distinctive difference between seq-shellability and bishellability is that the latter demands a symmetric structure in the sense that we can mutually exchange the sequences $a_0, \ldots, a_k$ and $b_0, \ldots, b_k$. Thus, the sequence $b_0, \ldots, b_{k-i}$ has to be the simple sequence of a_i in the subdrawing $D - \{a_0, \ldots, a_{i-1}\}$ for all $0 \leq i \leq k$ and vice versa, i.e. the sequence $a_0, \ldots, a_{k-i}$ has to be the simple sequence of b_i in the subdrawing $D - \{b_0, \ldots, b_{i-1}\}$ for all $0 \leq i \leq k$. With seq-shellability we do not have this requirement. Here we have the vertex sequence $a_0, \ldots, a_k$ and each vertex a_i with $0 \leq i \leq k$ has its own (independent) simple sequence S_i.

Figure 5 shows a gadget that visualizes the difference between bishellability and seq-shellability: (a) shows a substructure with nine vertices that may occur in a drawing. We have the simple sequence v_1, v_2, v_4 for vertex v_3 in (b) and (c). Therefore, we can remove vertex v_3 and are able to guarantee the

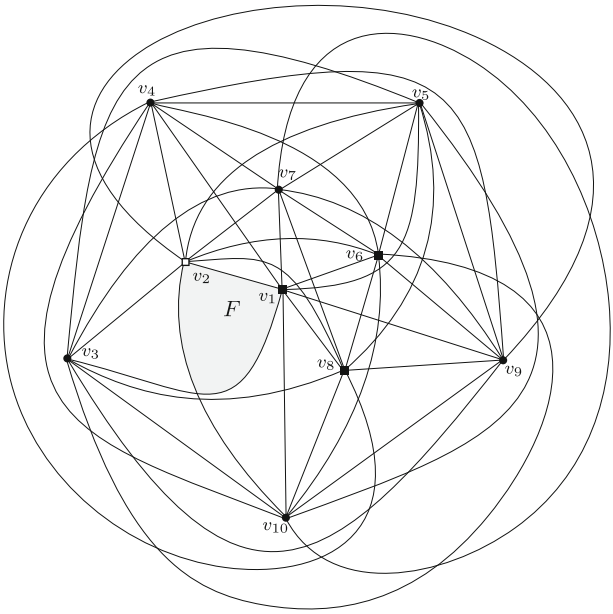


Fig. 4. Subdrawing $H - v_0$ after removing vertex v_0 and its incident edges. The second vertex of the vertex sequence v_2 is incident to the face containing F and has simple sequence S_1. Vertex v_2 and the vertices of S_1 are highlighted as unfilled and filled squares.

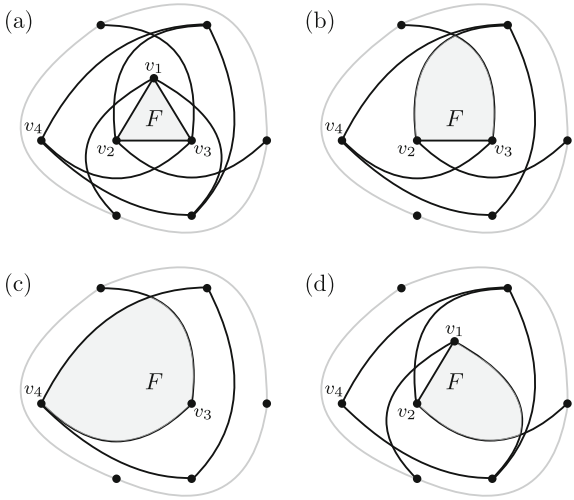


Fig. 5. The gadget does not allow for a bishellability sequence, because only one of the two sequences $a_0, \ldots, a_k$ or $b_0, \ldots, b_k$ can be chosen due to condition three of the definition of bishellability. However, the gadget is seq-shellable.

number of invariant edges. After removing vertex v_3 in (d), there are simple sequences for vertex v_1 and v_2, thus the substructure is seq-shellable. However, it is impossible to apply the definition of bishellability. We may use, for example, sequence v_1, v_2, v_4 as $a_0, \ldots, a_k$ sequence and we need a second sequence (the b sequence) that satisfies the exclusion condition of the bishellability, i.e. for each $i \in \{0, \ldots, k\}$, the set $\{a_0, a_1, \ldots a_i\} \cap \{b_{k-i}, b_{k-i-1}, \ldots, b_0\}$ has to be empty (see Definition 1). The first vertex of our second sequence (i.e. b_0) has to be v_3, because b_0 has to be incident to F. Now, for the second vertex we have to satisfy $\{a_0, a_1\} \cap \{b_1, b_0\} = \emptyset$, thus the second vertex has to be different from the first two vertices of the sequence v_1, v_2, v_4. Because we only can choose between vertices v_1 and v_2, we cannot select a second vertex for our b sequence. Thus, the structure is not bishellable. We can argue the same way for the other possible sequences in the gadget.

4 Conclusion

In this work, we introduced the new class of seq-shellable drawings and verified the Harary-Hill conjecture for this class. Seq-shellability is a generalization of bishellability, thus bishellability implies seq-shellability. In addition we exhibited a drawing of K_{11} which is seq-shellable but not bishellable, hence seq-shellability is a proper extension of bishellability. So far, we are not aware of an optimal seq-shellable but non-bishellable drawing and we close with the following open questions:

1. Can we find a construction method to obtain optimal drawings of K_n that are seq-shellable but not bishellable?
2. Does there exist a non-bishellable but seq-shellable drawing of K_n with $10 \leq n < 14$, such that after removing the first vertex of the simple sequence the drawing $D - a_0$ is still non-bishellable. We found a drawing of K_{14} with this property.

References

1. Guy, R.K.: A combinatorial problem. Nabla Bull. Malay. Math. Soc. **7**, 68–72 (1960)
2. Harary, F., Hill, A.: On the number of crossings in a complete graph. Proc. Edinb. Math. Soc. **13**(4), 333–338 (1963)
3. Pan, S., Richter, R.B.: The crossing number of K_{11} is 100. J. Graph Theory **56**(2), 128–134 (2007)
4. McQuillan, D., Pan, S., Richter, R.B.: On the crossing number of K_{13}. J. Comb. Theory Ser. B **115**, 224–235 (2015)
5. Ábrego, B., Aichholzer, O., Fernández-Merchant, S., Hackl, T., Pammer, J., Pilz, A., Ramos, P., Salazar, G., Vogtenhuber, B.: All good drawings of small complete graphs. In: Proceedings of 31st European Workshop on Computational Geometry (EuroCG), pp. 57–60 (2015)
6. Ábrego, B., Aichholzer, O., Fernández-Merchant, S., Ramos, P., Salazar, G.: The 2-page crossing number of K_n. In: Proceedings of the Twenty-Eighth Annual Symposium on Computational Geometry. SoCG 2012, pp. 397–404. ACM, New York (2012)
7. Ábrego, B.M., Aichholzer, O., Fernández-Merchant, S., Ramos, P., Salazar, G.: Shellable drawings and the cylindrical crossing number of K_n. Discrete Computa. Geom. **52**(4), 743–753 (2014)
8. Balko, M., Fulek, R., Kyncl, J.: Crossing numbers and combinatorial characterization of monotone drawings of K_n. Discrete Computa. Geom. **53**(1), 107–143 (2015)
9. Ábrego, B.M., Aichholzer, O., Fernández-Merchant, S., Ramos, P., Salazar, G.: More on the crossing number of K_n: monotone drawings. Electron Notes Discrete Math. **44**, 411–414 (2013)
10. Ábrego, B., Aichholzer, O., Fernández-Merchant, S., McQuillan, D., Mohar, B., Mutzel, P., Ramos, P., Richter, R., Vogtenhuber, B.: Bishellable drawings of K_n. In: Proceedings of XVII Encuentros de Geometría Computacional (EGC), Alicante, Spain, pp. 17–20 (2017)
11. Székely, L.A.: A successful concept for measuring non-planarity of graphs: the crossing number. Electron. Notes Discrete Math. **5**, 284–287 (2000)
12. Schaefer, M.: The graph crossing number and its variants: a survey. Electron. J. Comb. **1000**, DS21 (2013)
13. Ábrego, B.M., Cetina, M., Fernández-Merchant, S., Leaños, J., Salazar, G.: On $\leq k$-edges, crossings, and halving lines of geometric drawings of K_n. Discrete Comput. Geom. **48**(1), 192–215 (2012)
14. Erdös, P., Lovász, L., Simmons, A., Straus, E.G.: Dissection graphs of planar point sets. In: Srivastava, J.N. (ed.) A Survey of Combinatorial Theory, pp. 139–149. Elsevier, Amsterdam (1973)
15. Lovász, L., Vesztergombi, K., Wagner, U., Welzl, E.: Convex quadrilaterals and k-sets. Contemp. Math. **342**, 139–148 (2004)

Cryptographic Limitations on Polynomial-Time *a Posteriori* Query Learning

Mikito Nanashima(✉)

Department of Mathematical and Computing Science,
Tokyo Institute of Technology,
O-okayama 2-12-1, Meguro-ku, Tokyo 152-8552, Japan
nanashima.m.aa@is.c.titech.ac.jp

Abstract. We investigate the polynomial-time learnability by using examples and membership queries. Angluin and Kharitonov [1] proved that various concept classes (e.g., Boolean formulae, non-deterministic finite automata) are not polynomial-time learnable in this learning model based on a public-key encryption scheme with a certain security (i.e., IND-CCA1). As a stronger learning model, we consider an *a posteriori* query learning model, and show that it is indeed stronger than the above learning model if a one-way function exists. Nevertheless, from a secure encryption scheme, we prove that many natural classes containing Boolean formula concept class is not polynomial-time learnable even in this stronger learning model. The security of an encryption scheme used in this paper is weaker than the one used by Angluin and Kharitonov.

Keywords: Computational learning theory · PAC learning
Query learning · Cryptography · Encryption · Signature

1 Introduction

Computational learning theory introduced by Valiant [10] is concerned with what machines can learn efficiently. Roughly speaking, a learning algorithm tries to learn a certain target concept such as an automaton or a Boolean formula, and we say that a concept class (i.e., a set of concepts) is polynomial-time learnable if a polynomial-time learning algorithm can learn any concept in the class approximately correctly under any example distribution by examples and membership queries. There are a number of studies and many positive results and negative results are known. On the negative side, it is known that cryptographic tools and assumptions are useful to show the limits of polynomial-time learnability. For example, from well-known cryptographic assumptions (e.g., hardness of inverting the RSA function, factoring Blum integers) Angluin and Kharitonov show that many concept classes (e.g., Boolean formulae, non-deterministic finite automata) are not polynomial-time learnable. In general, their argument is based on an

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 285–297, 2018.
https://doi.org/10.1007/978-3-319-94667-2_24

a priori chosen-ciphertext secure public-key encryption scheme (IND-CCA1) constructed from the cryptographic assumptions [1].

We consider a stronger learning model in this paper. In the above model, learning can be divided into two phases: a learning phase and a testing phase [6]. In a learning phase, an algorithm learns a target concept by asking queries to a teacher, called an "oracle", who knows the target concept. In a testing phase (after the learning phase), a learner is given a test, called a challenge, by the oracle and answers it correctly to prove that the learner has learned the target concept successfully. In the testing phase, the learning algorithm cannot make a query to the oracle. Let us call this *a priori* query learning. In this paper we introduce a new learning model—*a posteriori* query learning—that is stronger in the sense that a learner is allowed to query to the oracle while in a testing phase (like an open-book examination). We can easily show that there is no difference between *a priori* and *a posteriori* query learning in the restricted case where only examples are allowed. On the other hand, in the case of learning with membership queries (i.e., when a learning algorithm is allowed to ask the teacher whether a certain object (except the challenge) is contained in the target concept or not), we show that *a posteriori* query learning is indeed stronger than *a priori* one if a one-way function exists.

We study in this paper whether natural concept classes such as Boolean formulae are polynomial-time learnable in this *a posteriori* query learning. Note that it is not so difficult to prove the non-learnability result similar to the one by Angluin and Kharitonov by using an encryption scheme with strong security (i.e., IND-CCA2) and the decryption method computable in a certain complexity class. Our main result is to give a concept class that is not *a posteriori* query learnable in polynomial-time based on a standard one-way function. This leads to the non-learnability of natural concept classes based on an encryption scheme with weaker security, that is, indistinguishable encryptions for multiple messages (where we also require that the scheme has a decryption method computable in a certain complexity class). Note that it has been shown that an encryption scheme with indistinguishable encryptions for multiple messages is in general weaker than the IND-CCA1 encryption scheme [2]. Thus, we derived a stronger non-learnability result from a weaker cryptographic assumption.

We also investigate polynomial-time learnability by using only membership queries (i.e., no examples). In this restricted case, we give a non-learnable concept class based on an encryption scheme with single-massage security. Note that it is weaker than multiple-messages security and we can construct the weak encryption scheme unconditionally by the well-known construction (i.e., one-time pad). This leads to the non-learnability result of natural concept classes based on a standard one-way function without any condition for the computational complexity of cryptographic tools (e.g., an easily computable decryption).

The motivation of considering a stronger learning model is mainly to investigate how machines can learn complex concept classes efficiently. As we mentioned above, Angluin and Kharitonov showed that many natural classes are not polynomial-time learnable in *a priori* query learning under some cryptographic

assumptions [1]. Thus as a stronger model, we consider *a posteriori* query learning and investigate whether it is indeed useful to learn some of these natural classes. In some fields, especially cryptography, we may observe the situation that any membership query except the challenge is allowed. We can regard adversaries in cryptography as learning algorithms in learning theory. From the perspective of cryptography, for example, the aim of the learner trying to learn a secret-key is to obtain some secret information from a certain ciphertext (i.e., a challenge) rather than the secret-key itself, and the learner may get other information adaptively by abusing complicated applications using the secret-key internally. In other words, there is a gap between complex cryptographic security and non-learnability in the *a priori* query learning model. The characterization of cryptographic security by difficulty of learning is useful to understand the relationship between some cryptographic tools, and our motivation is also to reduce the gap between cryptography and computational learning theory.

2 Preliminaries

2.1 Concepts

We use a binary alphabet. If x is a string, $|x|$ denotes its length. We let $\langle x, y\rangle$ denote a proper encoding of a pair (x, y) of strings as a single string.

A representation of concepts (or a concept class) $\mathscr{C}$ is defined as any subset of $\{0,1\}^* \times \{0,1\}^*$. For an element $(u, x) \in \mathscr{C}$, we call the first string u a concept name, and the second string x an example. We define a concept represented by a concept name u as a set of strings $\kappa_{\mathscr{C}}(u) = \{x : (u, x) \in \mathscr{C}\}$. We also define a computational complexity of a representation of concepts $\mathscr{C}$ as the complexity of deciding whether $(u, x) \in \mathscr{C}$ or not.

Now we give some examples. We let $\mathscr{C}_{BF}$ denote the representation of concepts for Boolean formulae. Then (u, x) is an element of $\mathscr{C}_{BF}$ if and only if u has the form $\langle n, \bar{\phi}\rangle$, where n is an integer and $\bar{\phi}$ is a proper encoding of Boolean formula ϕ over n variables, and x is a string satisfying $|x| = n$ and $\phi(x) = 1$. For example, the concept represented by the concept name $u = \langle 3, \neg x_1 \vee (x_2 \wedge x_3)\rangle$ is $\kappa_{\mathscr{C}_{BF}}(u) = \{000, 001, 010, 011, 111\}$.

In this paper, we refer to the classes[1] NC^i and AC^i together as a circuit complexity class. For brevity, we may regard the circuit complexity class as a set of circuits. For a circuit complexity class $\mathscr{S}$, we let $\mathscr{C}_{\mathscr{S}}$ denote the representation of concepts for circuits in $\mathscr{S}$. Then (u, x) is an element of $\mathscr{C}_{\mathscr{S}}$ if and only if u has the form $\langle n, \bar{c}\rangle$, where n is a integer and $\bar{c}$ is a proper encoding of a circuit c in $\mathscr{S}$ that has n input-gates and 1 output-gate, and x is a string satisfying $|x| = n$ and $c(x) = 1$.

[1] Let n be the length of input. NC^i is the class computed by a family of polynomial size and $O(\log^i(n))$ depth circuits with *bounded* fan-in, and AC^i is the similar class except that gates are allowed to have *unbounded* fan-in.

2.2 Learning Models

We generalize the definitions in [1,6]. Let $\mathscr{C}$ be a representation of concepts. A learning algorithm A for $\mathscr{C}$ is defined as any randomized oracle Turing machine that takes as input a bound s on the length of the target concept name, a bound n on the length of examples, and an accuracy parameter $\epsilon \in (0, 1/2]$. The algorithm A can make a call to the CHALLENGE oracle at only one time, and the oracle returns a string $\tilde{x}$, called a challenge, chosen according to unknown example distribution D on $\{0,1\}^{\leq n}$. The goal of A is to guess and output whether the challenge $\tilde{x}$ is contained in the unknown target concept represented by $u \in \{0,1\}^{\leq s}$ correctly with probability at least $1-\epsilon$. The algorithm A can also make a call to the following oracles decided by the target concept u and the example distribution D to gain knowledge about the target concept.

- EXAMPLE: the oracle takes no input and returns a pair (x, b), where x is a string chosen according to D and $b = 1$ if $x \in \kappa_{\mathscr{C}}(u)$ and $b = 0$ otherwise.
- MQ (Membership Query): the oracle takes a string $x \in \{0,1\}^*$ as input and returns 1 if $x \in \kappa_{\mathscr{C}}(u)$ and 0 otherwise.

We classify learning algorithms under the accessibility to the oracles as follows:

- *a priori* query learning: it can make a call to EXAMPLE and MQ only before receiving a challenge.
- *a posteriori* query learning: it can also make a call to EXAMPLE and MQ after receiving a challenge, but querying the challenge to MQ is prohibited.

We say that a representation of concepts $\mathscr{C}$ is polynomial-time learnable if there exists a learning algorithm A satisfying the following conditions: for all inputs (s, n, ϵ), concept names $u \in \{0,1\}^{\leq s}$, and distributions D on $\{0,1\}^{\leq n}$, (1) A runs in polynomial-time in s, n, and $1/\epsilon$, and (2) A succeeds in predicting whether the challenge is in the target concept $\kappa_{\mathscr{C}}(u)$ under the distribution D correctly with probability at least $1-\epsilon$.

We also define weaker learnability of a concept class. A weak learning algorithm is a randomized oracle machine with the same properties as the above learning algorithm except for taking no accuracy parameter. It tries to predict a correct classification of a challenge with probability more than $1/2$ (in the case of making a prediction at random). We say that a representation of concepts $\mathscr{C}$ is weakly polynomial-time learnable if there exists a weak learning algorithm A and a polynomial p satisfying the following conditions: for all inputs (s, n), concept names $u \in \{0,1\}^{\leq s}$, and distributions D on $\{0,1\}^{\leq n}$, (1) A runs in polynomial-time in s and n, and (2) A succeeds in predicting whether the challenge is in the target concept $\kappa_{\mathscr{C}}(u)$ under the distribution D correctly with probability at least $1/2 + 1/p(s, n)$.

It is known that polynomial-time learnability in *a priori* query learning is equivalent to the one in PAC (Probably Approximately Correct) learning introduced by Valiant [4,10]. In the PAC learning model, it is known that weak polynomial-time learnability is equivalent to (strong) polynomial-time learnability [8]. Thus, the relationship also holds in an *a priori* query learning model.

Lemma 1 ([8]). *In a priori query learning by only examples, weak polynomial-time learnability is equivalent to polynomial-time learnability.*

In the restricted case where only examples are allowed, an *a priori* query learner can simulate an *a posteriori* query learner by getting enough examples in advance. Then, the following holds. (We omit the proof.)

Lemma 2. *In polynomial-time learning by only examples, a priori query learnability is equivalent to a posteriori query learnability. In weak polynomial-time learnability, the same result holds.*

By the above lemmas, we have immediately the following lemma.

Lemma 3. *In a posteriori query learning by only examples, weak polynomial-time learnability is equivalent to polynomial-time learnability.*

2.3 Cryptographic Tools

We introduce some useful cryptographic tools to show the limits of polynomial-time learnability.

Definition 1 (one-way function). *A function f is one-way if f is polynomial-time computable but any non-uniform polynomial-time algorithm cannot invert f.*

Definition 2 (encryption scheme). *An encryption scheme is a triple (G, E, D) of PPT (probabilistic polynomial-time) algorithms with the following properties: (1) for every integer n, $G(1^n)$ outputs a pair (e, d) of strings, where e is an encryption-key and d is a decryption-key, (2) for every pair (e, d) of keys and plaintext $\alpha \in \{0,1\}^*$, the algorithms E and D satisfy the condition that $\Pr[D(d, E(e, \alpha)) = \alpha] = 1$.*

Definition 3 (signature scheme). *A signature scheme is a triple (G, S, V) of PPT algorithms with the following properties: (1) for every integer n, $G(1^n)$ outputs a pair (sk, vk) of strings, where sk is a signing-key and vk is a verification-key, (2) for every pair (sk, vk) of keys and document $\alpha \in \{0,1\}^*$, the algorithms S and V satisfy the condition that $\Pr[V(vk, \alpha, S(sk, \alpha)) = 1] = 1$.*

In this paper, we will make no distinction between a private-key encryption scheme and a public-key encryption scheme. Besides, we only consider a deterministic algorithm as a decryption algorithm D and a verification algorithm V. We let $G_1(1^n)$ (resp. $G_2(1^n)$) denote the first (resp. second) element of outputs of $G(1^n)$ in both cases of an encryption scheme and a signature scheme.

3 Polynomial-Time *a Posteriori* Query Learnability

We discuss polynomial-time learnability in an *a posteriori* query learning model with membership queries.

3.1 Polynomial Representation of Signatures

To show that membership queries don't help to learn some natural concept classes, we introduce a useful tool, which is polynomial representation of signatures by Angluin and Kharitonov [1].

Definition 4 ([1]). *Let $\mathscr{C}$ and $\mathscr{C}'$ be representations of concepts. $\mathscr{C}'$ polynomially represents signatures for $\mathscr{C}$ with respect to a signature scheme (G, S, V) if there exist two maps, a concept map g and an example map f, and two polynomials q_1 and q_2 such that for all integers s and n, concept names $u \in \{0,1\}^{\leq s}$, examples $x \in \{0,1\}^{\leq n}$, pairs of keys $(sk, vk) \leftarrow G(1^n)$, and signatures $y \leftarrow S(sk, x)$, the following conditions hold:*

(1) $|g(vk, s, n, u)| \leq q_1(s, n)$,
(2) $w := f(vk, s, n, x, y)$ *is computable in polynomial-time in* s *and* n, *and* $|w| \leq q_2(s, n)$,
(3) $f_{vk,s,n}(x, y) := f(vk, s, n, x, y)$ *is injective, and there is a polynomial-time algorithm that computes the unique inverse element (if any),*

and for all integers s and n, concept names $u \in \{0,1\}^{\leq s}$, pairs of keys $(sk, vk) \leftarrow G(1^n)$, and strings $w \in \{0,1\}^*$, the following condition holds:

(4) $w \in \kappa_{\mathscr{C}'}(g(vk, s, n, u))$ *if and only if there exist strings x and y such that* $w = f_{vk,s,n}(x, y)$ *and* $x \in \kappa_{\mathscr{C}}(u)$ *and* $V(vk, x, y) = 1$.

Intuitively, $\mathscr{C}$ and $\mathscr{C}'$ satisfy the above conditions if each concept in $\mathscr{C}'$ corresponds to each concept in $\mathscr{C}$ with valid signatures to the examples. Angluin and Kharitonov proved that many natural concept classes polynomially represent signatures for themselves with respect to any signature scheme. Their proof holds as long as the concept class is closed by AND ($\wedge$) and has ability to compute 3CNF formulae. It is easily checked that any 3CNF formula can be computed by a depth-3 circuit using unbounded fan-in gates. Thus, we have the following theorem.

Theorem 1 ([1]). *$\mathscr{C}_{BF}$ polynomially represents signatures for $\mathscr{C}_{BF}$ with respect to every signature scheme. Moreover, $\mathscr{C}_{\mathscr{S}}$ polynomially represents signatures for $\mathscr{C}_{\mathscr{S}}$ with respect to every signature scheme, where $\mathscr{S}$ is a circuit complexity class that contains AC^0.*

Angluin and Kharitonov also showed that membership queries don't help to learn a concept class that polynomially represents the signatures with the following security, called EUF-CMA. It is well-known that an EUF-CMA signature scheme can be constructed from any one-way function [7].

Definition 5 (EUF-CMA [3]). *A signature scheme (G, S, V) is existential unforgeable against adaptive chosen message attackers (EUF-CMA) if for all non-uniform oracle machines M, polynomials p, and sufficiently large integers n, the following condition holds:*

$$\Pr\left[\begin{array}{l} V(vk, \alpha, \beta) = 1 \wedge \alpha \notin Q_M^{S_{sk}}(vk) \\ \quad \textit{where } (sk, vk) \leftarrow G(1^n) \textit{ and } (\alpha, \beta) \leftarrow M^{S_{sk}}(vk) \end{array}\right] < \frac{1}{p(n)},$$

where, $Q_M^{S_{sk}}(vk)$ *denotes a set of* M*'s queries to a signing oracle* $S_{sk}(\cdot)$*.*

Theorem 2 ([1]). *Let* $\mathscr{C}$ *and* $\mathscr{C}'$ *be concept classes in* P*. If* $\mathscr{C}'$ *polynomially represents signatures for* $\mathscr{C}$ *with respect to an EUF-CMA signature scheme and* $\mathscr{C}$ *is not polynomial-time learnable by only examples, then* $\mathscr{C}'$ *is not polynomial-time learnable by examples and membership queries in a priori query learning.*

3.2 Positive Results

On the assumption that a one-way function exists, we show that *a posteriori* query learning has truly stronger power than *a priori* query learning in the case where examples and membership queries are available.

From an encryption scheme that has indistinguishable encryptions for multiple messages, we first show how to construct a concept class that is not polynomial-time learnable by only examples. Note that a private-key encryption scheme with this security can be constructed from any one-way function [2].

Definition 6 (indistinguishable encryptions for multiple messages [2]). *An encryption scheme* (G, E, D) *has indistinguishable encryptions for multiple messages if for any polynomial* t*, plaintexts* $\bar{x} = (x_1, \ldots, x_{t(n)})$*,* $\bar{y} = (y_1, \ldots, y_{t(n)})$ *(with* $|x_1| = |y_1| = \cdots = |x_{t(n)}| = |y_{t(n)}| = poly(n)$*), non-uniform polynomial-time algorithm* A*, polynomial* p*, and sufficiently large integer* n*, the following condition holds:*

$$|\Pr[A(\bar{E}_{G_1(1^n)}(\bar{x}), (G_1(1^n))) = 1] - \Pr[A(\bar{E}_{G_1(1^n)}(\bar{y}), (G_1(1^n))) = 1]| < \frac{1}{p(n)},$$

where, $\bar{E}_e(\bar{x})$ *denotes the sequence of the ciphertexts of* $x_1, \ldots, x_{t(n)}$ *by using the encryption-key* e *and independently tossed internal coins for each encryption process. An encryption-key* $G_1(1^n)$ *is only given to* A *in the public-key setting.*

Lemma 4. *If there exists an encryption scheme* (G, E, D) *that has indistinguishable encryptions for multiple messages and the decryption* D *is computable in a complexity class* $\mathscr{S}$*, then there exists a representation of concepts* $\mathscr{C}_D$ *in* $\mathscr{S}$ *that is not polynomial-time a posteriori query learnable by only examples.*

Proof. Define a representation of concepts $\mathscr{C}_D$ as follows:

$$\mathscr{C}_D = \{(d, x) : n \in \mathbb{N}, d \leftarrow G_2(1^n), x \in \{0,1\}^* \ s.t.\ D(d, x) = 1\}.$$

To determine whether $(d, x) \in \mathscr{C}_D$ or not, we have only to execute $D(d, x)$, hence $\mathscr{C}_D$ is in $\mathscr{S}$. Assume by contradiction that $\mathscr{C}_D$ is polynomial-time *a posteriori* query learnable by examples and let A be the learning algorithm and p be the polynomial that bounds on A's running time. For input 1^n to the key generator, let q_1 be a polynomial in n that bounds on the length of decryption-key and q_2 be a polynomial in n that bounds on the length of a ciphertext of a 1-bit plaintext.

We also define a polynomial t as $t(n) := p(q_1(n), q_2(n))$. For $0 \leq i \leq 2t(n) + 1$, assign the value to plaintexts x_i and y_i as follows:

$$x_0 = 1,\ y_0 = 0,\ x_1 = 0,\ y_1 = 1,\ x_i = y_i = i \mod 2\ (i > 1),$$

We construct a distinguisher B for the multiple messages $\bar{x} = (x_0, \ldots, x_{2t(n)+1})$, and $\bar{y} = (y_0, \ldots, y_{2t(n)+1})$ by using A. For the details, see the construction of the algorithm B.

Algorithm B

Input: $c_0, \cdots, c_{2t(n)+1}$

1: $i := 1$

2: Execute $A(q_1(n), q_2(n), 1/4)$, where answer to A's queries as follows:
EXAMPLE: Select $r \in \{0, 1\}$ uniformly at random, and return c_{2i+r} to A. Then, set $i := i + 1$.
CHALLENGE: Select $\tilde{r} \in \{0, 1\}$ uniformly at random, and return $c_{\tilde{r}}$ as A's challenge.

3: If A halts and outputs some prediction b, then B outputs $b \oplus \tilde{r}$.

Let (e, d) be the pair of keys firstly selected, then it can be checked that B runs in polynomial-time in n and perfectly simulates A's oracles decided by the distribution $E_e(U_1)$ and the target concept $\kappa_{\mathscr{C}_D}(d)$, where U_1 denotes the uniform distribution over $\{0, 1\}$. When input to B is taken from $E_e(\tilde{x})$, the probability that B outputs 1 is equal to the success probability of A, which is at least $3/4$. In another case that input is taken from $E_e(\tilde{y})$, the probability that B outputs 1 is equal to the error probability of A, which is at most $1/4$. Therefore, B distinguishes two ciphertexts with probability at least $3/4 - 1/4 = 1/2$. This contradicts the security of the encryption scheme. □

Now we construct the witness for the gap between *a priori* query learning and *a posteriori* query learning by using this concept class.

Theorem 3. *If a one-way function exists, there exists a representation of concepts that is not polynomial-time a priori query learnable but polynomial-time a posteriori query learnable by examples and membership queries.*

Proof (sketch). Let (G_e, E, D) be a private-key encryption scheme with indistinguishable encryptions for multiple messages, which can be constructed from any one-way function [2]. By Lemma 4, $\mathscr{C}_D$ is not polynomial-time *a priori* query learnable by examples. Besides, let (G_s, S, V) be an EUF-CMA signature that has a polynomial-time computable function g with $V(vk, \alpha, \beta) = 1 \Leftrightarrow V(vk, \alpha, g(\beta)) = 1$ for any verification-key vk. (Firstly we construct a general EUF-CMA signature scheme from a one-way function, and let the signing algorithm concatenate a signature with an extra random bit. We let the verification

algorithm ignore the extra least significant bit (LSB), and define g as a function that negates a LSB.) Now we define a representation of concepts $\mathscr{C}$ as follows:

$$\mathscr{C} = \{(\langle d, vk\rangle, \langle x, y\rangle) : n_1, n_2 \in \mathbb{N}, d \leftarrow G_{e2}(1^{n_1}), vk \leftarrow G_{s2}(1^{n_2}), \\ x, y \in \{0,1\}^* \; s.t. \; D(d,x) = 1, V(vk,x,y) = 1\}.$$

Then, it can be checked that $\mathscr{C}$ polynomially represents signatures for $\mathscr{C}_D$ with respect to (G_s, S, V). Hence by Theorem 2, $\mathscr{C}$ is not polynomial-time *a priori* learnable by examples and membership queries. On the other hand, we construct an *a posteriori* query learner A. For the details, see the construction of the algorithm A. When A learns a target concept $\kappa_{\mathscr{C}}(\langle d, vk\rangle)$, we have the following.

Algorithm A

Input: (s, n, ϵ)

1: Request a challenge w
2: Check that w is in the form $\langle x, y\rangle$ (if not, A outputs 0 and halts).
3: Query $\langle x, g(y)\rangle$ to MQ, and output the answer.

$$\mathrm{MQ}(\langle x, g(y)\rangle) = 1 \Leftrightarrow V(vk, x, g(y)) = 1 \wedge D(d,x) = 1 \\ \Leftrightarrow V(vk,x,y) = 1 \wedge D(d,x) = 1 \Leftrightarrow w = \langle x, y\rangle \in \kappa_{\mathscr{C}}(\langle d, vk\rangle).$$

Hence, the *a posteriori* query learner A learns $\mathscr{C}$ successfully with probability 1 under any example distribution. □

3.3 Negative Results

By Theorem 3, we showed *a posteriori* query learning has indeed stronger power than *a priori* one. However, the witness was not natural at all, and it is not known that there is a natural witness to the gap. As a negative answer to this, we show *a posteriori* query learning has no additional power about many natural concept classes.

The main idea is to use a signature scheme with stronger security, called sEUF-CMA, in the proof of Theorem 2 to show the same result in *a posteriori* query learning. Note that we can also construct an sEUF-CMA signature scheme from any one-way function [2].

Definition 7 (sEUF-CMA [9]). *A signature scheme (G, S, V) is strongly existential unforgeable against adaptive chosen message attackers (sEUF-CMA) if for all non-uniform oracle machines M, polynomials p, and sufficiently large integers n, the following condition holds:*

$$\Pr\left[\begin{array}{l} V(vk,\alpha,\beta) = 1 \wedge (\alpha,\beta) \notin QA_M^{S_{sk}}(vk) \\ \textit{where } (sk, vk) \leftarrow G(1^n) \textit{ and } (\alpha,\beta) \leftarrow M^{S_{sk}}(vk) \end{array}\right] < \frac{1}{p(n)},$$

where, $QA_M^{S_{sk}}(vk)$ denotes a set of pairs of M's query to the signing oracle $S_{sk}(\cdot)$ and the answer to the query (that is, pairs of a message and a valid signature).

Theorem 4. *Let $\mathscr{C}$ and $\mathscr{C}'$ be representations of concepts in P. If $\mathscr{C}'$ polynomially represents signatures for $\mathscr{C}$ with respect to an sEUF-CMA signature scheme (G, S, V) and $\mathscr{C}$ is not polynomial-time learnable by only examples, then $\mathscr{C}'$ is not polynomial-time learnable by examples and membership queries in a posteriori query learning.*

Proof. Assume $\mathscr{C}'$ is polynomial-time *a posteriori* query learnable by examples and membership queries, and let A' be the learning algorithm. By Lemma 3, we only need to show $\mathscr{C}$ is weakly polynomial-time learnable by examples. Since $\mathscr{C}'$ polynomially represents signatures for $\mathscr{C}$, we have the mappings g, f and the polynomials q_1, q_2 satisfying the conditions in Definition 4. Then, we construct a weak learning algorithm A that learns $\mathscr{C}$ with probability at least $3/4$ by using A' as follows:

Algorithm A

Input: (s, n)
Oracle: $\mathrm{EXAMPLE}_A$, $\mathrm{CHALLENGE}_A$
1: Generate $(sk, vk) \leftarrow G(1^n)$.
2: Execute $A'(q_1(n), q_2(n), 1/8)$, where answer to A''s queries as follows:
$\mathrm{EXAMPLE}_{A'}$: Have access to $\mathrm{EXAMPLE}_A$ to get an example (x, b), and save it in memory. Then, A generates $y \leftarrow S_{sk}(x)$ and returns $(f_{vk,s,n}(x, y), b)$ to A'.
$\mathrm{CHALLENGE}_{A'}$: Have access to $\mathrm{CHALLENGE}_A$ to get challenge $\tilde{x}$. Then, A generates $\tilde{y} \leftarrow S_{sk}(\tilde{x})$ and returns $f_{vk,s,n}(\tilde{x}, \tilde{y})$ to A' as a challenge.
$\mathrm{MQ}_{A'}(w)$: (1) Compute (x, y) that satisfies $w = f_{vk,s,n}(x, y)$ (if not, return 0). (2) Check that $V(vk, x, y) = 1$ (if not, return 0). (3) Check whether (x, b) is in memory, if any, return b to A'. (4) Otherwise, return 0 to A' (for now).
3: If A' halts and outputs some prediction, then A also outputs the same prediction.

When A learns a target concept $\kappa_{\mathscr{C}}(u)$ under a distribution D, it can be checked that A runs in polynomial-time in s and n, and perfectly simulates A''s EXAMPLE and CHALLENGE oracles decided by the distribution $f_{vk,s,n}(D, S_{sk}(D))$ and the target concept represented by $g(vk, s, n, u)$ by the conditions in Definition 4. On the other hand, A may fail to answer to A''s membership query only in the case (4). However, it is guaranteed that it does not occur by the security of signature schemes. We explain the reason in more detail.

Assume that the case (4) occurs at the membership query $w = f_{vk,s,n}(x, y)$, then y is a valid signature to x because they satisfy $V(vk, x, y) = 1$ by (2). Besides, the signature y to x has not been generated yet because (x, b) does not exist in memory. (In the case that $x = \tilde{x}$, y is not equal to $\tilde{y}$ because A' cannot query the challenge $f_{vk,s,n}(\tilde{x}, \tilde{y})$ to MQ.) In other words, (x, y) is just a forged signature that breaks sEUF-CMA security.

Assume that the error probability of A is more than $1/4$. Then we construct an adversary M to the signature scheme by using A as follows: M takes a verification-key vk as an input and simulates A, where M generates signatures

by M's signing oracle. For infinitely many integer n's, M is given also the non-learnable concept name u_n and the worst examples from D_n for A beforehand as advice, and M lets A learn the target concept $\kappa_{\mathscr{C}}(u_n)$ under the distribution D_n. If the case (4) occurs in simulating A, M outputs a forged signature (x, y) as above. If it doesn't occur and A outputs some prediction, then M outputs "*error*" and halts. By the construction of A, if A fails to learn $\mathscr{C}$, then A fails to simulate A''s membership queries (in this case, M succeeds in generating a forged signature) or A' fails to learn $\mathscr{C}'$ under the correct oracle simulation. The latter probability is smaller than $1/8$ by the assumption about A'. Hence, M succeeds in outputting a forged signature with probability more than $1/4 - 1/8 = 1/8$, and this contradicts the sEUF-CMA security. Therefore, A weakly learns $\mathscr{C}$ with probability at least $3/4$. □

3.4 Learnability of Some Natural Classes

We apply the above lemmas and theorems to show non-learnability in some concrete concept classes from existence of an encryption scheme with weaker conditions than the previous work [1]. Firstly we have the following lemma by using Pitt and Warmuth's prediction-preserving reduction.

Lemma 5 ([6]). *If there exists polynomial-time unpredictable (i.e., not a priori learnable by examples) concept class $\mathscr{C}$ in NC^1, then $\mathscr{C}_{BF}$ is also polynomial-time unpredictable. Furthermore, if there exists a polynomial-time unpredictable concept class $\mathscr{C}$ in a circuit complexity class $\mathscr{S}$, then $\mathscr{C}_{\mathscr{S}}$ is also polynomial-time unpredictable.*

To use Theorem 4 in the following proof, we need to show that there exists a one-way function (which is equivalent to existence of an sEUF-CMA signature). In fact, an encryption scheme has indistinguishable encryptions for multiple messages implies the existence of a one-way function [5].

Theorem 5. *If there exists an encryption scheme that has indistinguishable encryptions for multiple messages and the decryption is computable in NC^1, then $\mathscr{C}_{BF}$ is not polynomial-time a posteriori query learnable by examples and membership queries.*

Proof. By the assumption and Lemma 4, there exists a polynomial-time unpredictable concept class $\mathscr{C}_D$ in NC^1. By Lemma 5, $\mathscr{C}_{BF}$ is also polynomial-time unpredictable. From the secure encryption scheme in the assumption, we construct an sEUF-CMA signature (G, S, V). By Theorem 1, $\mathscr{C}_{BF}$ polynomially represents signatures $\mathscr{C}_{BF}$ with respect to (G, S, V). Therefore, by Theorem 4, $\mathscr{C}_{BF}$ is not polynomial-time a posteriori query learnable by examples and membership queries. □

In general, we have the following theorem in the same way as Theorem 5. Note that we need the easier computable decryption with the multiple-messages security to prove the stronger limits of polynomial-time learnability.

Theorem 6. *Let $\mathscr{S}$ be a circuit complexity class that contains AC^0. If there exists an encryption scheme that has indistinguishable encryptions for multiple messages and the decryption is computable in $\mathscr{S}$, then $\mathscr{C}_{\mathscr{S}}$ is not polynomial-time a posteriori query learnable by examples and membership queries.*

3.5 Learnability with only Membership Queries

We discuss polynomial-time learnability in *a posteriori* query learning with only membership queries (i.e., no examples). In the proof of Lemma 4, multiple messages corresponded to multiple examples in learning. In other words, we can show polynomial-time non-learnability by no examples from an encryption scheme that has indistinguishable encryptions for a single message. Remember that we only used the security of the encryptions of 1-bit texts in the proof of Lemma 4, and we can easily construct a 1-bit private-key encryption scheme with this security by one-time pad. Thus, the following lemma holds. (We omit the proof.)

Lemma 6. *$\mathscr{C}_{OTP} = \{(0,1),(1,0)\}$ is not weakly polynomial-time learnable by no examples and no membership queries.*

We can convert a concept class that is not polynomial-time learnable by no examples and no membership queries into a concept class that is not polynomial-time *a posteriori* query learnable with only membership queries in the same way as Theorem 4, because examples were only used to simulate the example oracle in the proof. Hence, we have the following theorem.

Theorem 7. *Let $\mathscr{C}$ and $\mathscr{C}'$ be representations of concepts in P. If $\mathscr{C}'$ polynomially represents signatures for $\mathscr{C}$ with respect to an sEUF-CMA signature scheme and $\mathscr{C}$ is not weakly polynomial-time learnable by no examples, then $\mathscr{C}'$ is not polynomial-time a posteriori query learnable with only membership queries.*

Let $\mathscr{S}$ be a circuit complexity class that contains AC^0. It is obvious that $\mathscr{C}_{OTP}$ is in $\mathscr{S}$ because the evaluation problem can be solved by computing 1-bit parity. By using Pitt and Warmuth's prediction-preserving reduction, $\mathscr{C}_{\mathscr{S}}$ is also not weakly polynomial-time learnable by no examples. Therefore, we have the following result by Theorems 1 and 7.

Theorem 8. *Let $\mathscr{S}$ be a circuit complexity class that contains AC^0. If a one-way function exists, then $\mathscr{C}_{\mathscr{S}}$ is not polynomial-time a posteriori query learnable with only membership queries.*

References

1. Angluin, D., Kharitonov, M.: When won't membership queries help? J. Comput. Syst. Sci. **50**(2), 336–355 (1995)
2. Goldreich, O.: Foundations of Cryptography: Volume 2, Basic Applications. Cambridge University Press, New York (2004)

3. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. **17**(2), 281–308 (1988)
4. Haussler, D., Kearns, M., Littlestone, N., Warmuth, M.K.: Equivalence of models for polynomial learnability. In: Proceedings of the First Annual Workshop on Computational Learning Theory, pp. 42–55 (1988)
5. Impagliazzo, R., Luby, M.: One-way functions are essential for complexity based cryptography. In: Proceedings of the 30th Annual Symposium on Foundations of Computer Science, pp. 230–235 (1989)
6. Pitt, L., Warmuth, M.K.: Prediction-preserving reducibility. J. Comput. Syst. Sci. **41**(3), 430–467 (1990)
7. Rompel, J.: One-way functions are necessary and sufficient for secure signatures. In: Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing, pp. 387–394 (1990)
8. Schapire, R.E.: The strength of weak learnability. Mach. Learn. **5**(2), 197–227 (1990)
9. Stern, J., Pointcheval, D., Malone-Lee, J., Smart, N.P.: Flaws in applying proof methodologies to signature schemes. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 93–110. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_7
10. Valiant, L.G.: A theory of the learnable. Commun. ACM **27**(11), 1134–1142 (1984)

Placing Segments on Parallel Arcs

Yen Kaow Ng[1(✉)], Wenlong Jia[2], and Shuai Cheng Li[2]

[1] Department of Computer Science, Faculty of Information and Communication Technology, Universiti Tunku Abdul Rahman, Kampar, Malaysia
ykng@utar.edu.my
[2] Department of Computer Science, City University of Hong Kong, Kowloon Tong, Hong Kong
{wenlong,scli}@cityu.edu.hk

Abstract. In this paper we consider the problem of arranging segments on parallel arcs drawn within a circular sector, to provide foundational work for the visualization of genomic regions in the study of pathogenic integration. The arcs as well as the start and end angles for each segment are pre-defined; our problem is to place each segment on an arc without having them overlap. There are no segments that span multiple arcs. For visualization purpose, the segments are to be easily distinguishable. To achieve that we consider various criteria that in a sense, place segments as far as possible from each other—for instance, maximizing the sum of inter-center distances between nearest segments. We show complexity results for some of the resultant problems, while providing approximation or heuristic solutions for others. Our algorithms have been implemented in JavaScript and made available at https://github.com/kalngyk/segmentplacer.

Keywords: Segment placement · Approximation algorithm
Visualization

1 Introduction

In this paper we consider the problem of arranging segments on parallel *arcs*, or *curved tracks*, drawn within a circular sector (see Fig. 1). Our aim is to visualize genomic regions on circular sectors that are combined to form a disk, similar to those in Circos-like visualization [11,13].

The central angle, `maxangle`, of the sector is predetermined. Each segment is to start from a specified angle and end at a specified angle within the sector, but they can be placed on any track. Two segments that overlap in their specified angles must be placed on different tracks. There must be sufficiently many tracks such that segments do not overlap, but there cannot be too many tracks due to

1. Aesthetic reasons to avoid overly sparse placements, and
2. Pragmatic concerns of whether the resultant disk fits on an A4 or letter-sized paper (for printing as well as for inclusion in publications).

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 298–310, 2018.
https://doi.org/10.1007/978-3-319-94667-2_25

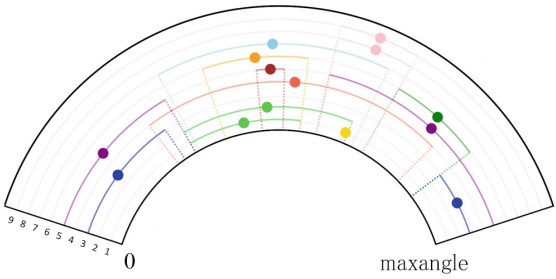


Fig. 1. Fourteen segments on nine tracks

The problem of finding such a non-overlapping placement resembles the interval scheduling problem [4] as well as a trivial variant of the offline dynamic storage allocation problem [5,6,9]. However, for our visualization purpose, a placement should ideally place segments far apart in order to maximize visual ease. This requirement gives rise to several non-trivial problems. In this paper we consider a few of these problems, and give foundational results regarding them. In particular, we consider situations where:

1. The distances between the nearest segments are in some sense maximized.
2. Segments are near-equally distributed on all the tracks.

The usage scenario which we have in mind is semi-automatic or interactive. The algorithm is to only assist by providing choices of placements, leaving the decision of what is aesthetically pleasing to the user. Hence, we look for algorithms that are: (1) sufficiently fast for interactive use, (2) able to enumerate all optimal solutions, (3) flexible in accommodating additional user requirements, such as requiring specific segments to be placed on given tracks.

1.1 Bioinformatics Use

The visualization we are aiming for is motivated by the study of genetic rearrangements due to tumor cells. It is known that tumor cells frequently harbor genome rearrangements or pathogenic integrations, such as oncogenic fusion-generating structure variations [7,10], chromothripsis [2], and oncovirus integrations [8,12]. The rearranged segments are often located in very specific sequences. Past studies into the the effects from these local genomic maps have revealed novel mechanisms in gene regulations [1,3,14].

Advances in biological sequencing technologies in recent years have furthered the accuracies in our identification of the affected regions, enabling more refined studies of the genomic regions. This increase in information, on the other hand, cannot be leveraged without accurate visualization of the resolved genomic allocations.

Current visualizations of genome rearrangements and oncovirus integrations are often presented using trajectory lines in Circos [11,13], which reveal little of

the segment relationships. By organizing genomic segments along parallel tracks, we hope to better display these relationships.

2 Preliminaries

In this section we give definitions that are shared throughout this paper. A *segment* represents an interval that is to be placed on an arc or track. The *locus* of a segment is a pair of positive integers (α, β) where $\alpha < \beta$. α and β span from 0 to `maxangle`. Intuitively, α indicates the start angle and β the end angle on the track where the segment is to be placed. For a segment s of locus (α, β), $\texttt{st}(s)$ is used to indicate α, $\texttt{ed}(s)$ used to indicate β, while $\texttt{ctr}(s)$ is used to indicate $(\beta + \alpha)/2$.

A *track* is a number from $\{1, 2, \ldots, k\}$. Tracks are considered to be possibly curved in this work, with track 1 having a radius of curvature r. Subsequent tracks $i > 1$ are separated by distance d; that is, they have radius of curvature $h(i) = r + (i - 1)d$.

A *placement* f of a set of segments $S = \{s_1, s_2, \ldots, s_n\}$, is a many-one relation where $f \subseteq \{(s, i) \mid s \in S, 1 \leq i \leq k\}$. A placement of two segments s and t, $\{(s, i), (t, j)\}$ say, is said to be *non-overlapping* iff $i \neq j$, or $\texttt{ed}(s) \leq \texttt{st}(t)$, or $\texttt{ed}(t) \leq \texttt{st}(s)$. A placement is non-overlapping iff every two segments in its domain are non-overlapping.

Given two elements (s, i), (s', i') in a placement, the shortest Euclidean distance between x and y, denoted $\texttt{dist}(x, y, i, i')$, for any $x \in \{\texttt{ed}(s), \texttt{st}(s), \texttt{ctr}(s)\}$ and $y \in \{\texttt{ed}(s'), \texttt{st}(s'), \texttt{ctr}(s')\}$ can be calculated from s, s', $h(i)$, and $h(i')$ with trigonometry.

2.1 Minimum Tracks Required for Non-overlapping Placement

Given a set of segments, the minimum number of tracks where a placement exists can be found using this simple approach from interval scheduling:

1. Order $s \in S$ by $\texttt{st}(s)$.
2. Set placement f to $\emptyset$.
3. For each $s \in S$ in increasing order of $\texttt{st}(s)$,
4. Find smallest $i \geq 0$ such that $f \cup \{(s, i)\}$ is non-overlapping.
5. Add pair (s, i) to f.

The largest i in the range of f is the minimum number of tracks required.

We can solve the problem similarly in the case that we require a minimum distance z between subsequent segments on the same track. More precisely, for two segments s and s' on the same track i, we want $\texttt{st}(s) - \texttt{ed}(s') \geq z$ if $\texttt{st}(s) \geq \texttt{ed}(s')$, and $\texttt{st}(s') - \texttt{ed}(s) \geq z$ if $\texttt{st}(s') \geq \texttt{ed}(s)$. This condition can be checked at line 4 in the program above.

Throughout the rest of this paper we assume that the number of tracks is predetermined by the user.

3 Maximize Within-Track Separation

We first consider the problem of maximizing the sum total of the separation between the segments that are placed on the same track. Maximizing this sum is tricky, because the problem is different when we consider whether to include the space:

(1) from 0 angle to the first segment, and
(2) from the end of the last segment to `maxangle`.

If they are considered, then the sum of all the spacing on all tracks would always be the same, except for differences due to the curvature of the arc. On the other hand, if we do not add these spaces to the sum, then a solution for the segments $(1, 2)$ and $(3, 4)$ on 2 tracks would (undesirably) place both segments on a single track, since placing the segments on separate tracks would result in zero separation score.

For this section we proceed with the inclusion of the distance from the start boundary of each track to its first segment, and ignore the curvature of the arc. (In the next section, we show an exhaustive search method that can be used to solve the case with curvature.)

For each placement f and each track i, let $\{s_1^{f,i}, s_2^{f,i}, \ldots, s_{n(f,i)}^{f,i}\}$ denote the segments placed by f on track i in increasing order of $\mathtt{st}(s)$, where $n(f, i)$ is the number of segments placed by f on track i, and let

$$\mathtt{within\_track\_space}(i) = \mathtt{st}(s_1^{f,i}) + \sum_{1 \le j < n(f,i)} \mathtt{st}(s_{j+1}^{f,i}) - \mathtt{ed}(s_j^{f,i}).$$

Then, our aim is to find f which maximizes $\sum_{1 \le i \le k} \mathtt{within\_track\_space}(i)$. This can be solved using a greedy algorithm as shown in Algorithm 1.

Algorithm 1. Algorithm to maximize $\sum_{1 \le i \le k} \mathtt{within\_track\_space}(i)$

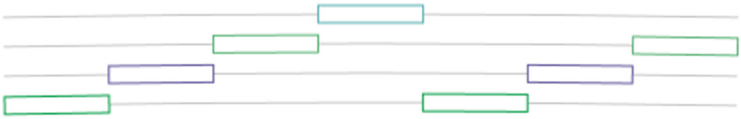
Input: Set of segments S, number of tracks k.
Output: Placement f maximizing $\sum_{1 \le i \le k} \mathtt{within\_track\_space}(i)$.

```
f ← ∅
for each track i do
    last_i ← NULL
for s ∈ S in increasing order of st(s) do
    j ← arg max_i (st(s) − max{ed(last_i), 0}).
    last_j ← s.
    Add (s, j) to f. Exit if f becomes overlapping.
return f.
```

To achieve optimality, it suffices that each track ends with a segment which ends some track in the optimal solution. That Algorithm 1 achieves optimality can be seen from the fact that it achieves this requirement. The output f can

also be shown to maximize $\min_{1\le i\le k, 1\le j<n(f,i)} \mathtt{st}(s_{j+1}^{f,i}) - \mathtt{ed}(s_j^{f,i})$. We omit the discussion from this manuscript.

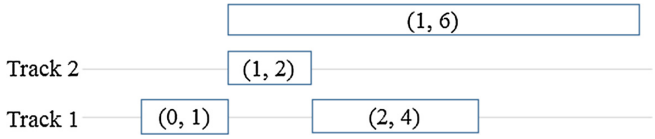
The algorithm is highly applicable, with a low runtime complexity of $O(nk)$, and produces reasonably placements in general. However, it has a tendency towards staircase-like placements such as the following.



4 Threshold Requirement to Inter-center Distances

In this section we consider the problem of requiring a threshold between the centers of every pair of segments. More precisely, we set a threshold z and require that for every s, $s' \in S$, $1 \le i, j \le k$, $\mathtt{dist}(\mathtt{ctr}(s), \mathtt{ctr}(s'), i, j) \ge z$.

This problem cannot be solved using a greedy algorithm where we consecutively place segments which are further away on the tracks, as we did for the earlier problem. For example, consider the instance where we are to place the segments $S = \{(0,1), (1,2), (2,4), (1,6)\}$ on 2 tracks of separation 1 with $z = 1.2$. The centers of the segments are respectively $\{0.5, 1.5, 3, 3.5\}$, and hence by the strategy should be placed in this order. A greedy algorithm similar to Algorithm 1 would place $(0,1)$ on track 1, $(1,2)$ on track 2, and then $(2,4)$ on track 1. However, that would make it impossible to place $(1,6)$; whereas if $(2,4)$ is placed on track 2 rather than track 1, it would be possible to place $(1,6)$.



An exhaustive search for all possible placements would on the other hand require $O(k^n)$ time. An alternative solution is to perform a search similar to the greedy strategy, but keeping all relevant "configurations" from earlier placements for a more complete search. More precisely, we place segments in increasing order of their centers. When placing each segment s, we examine all the possible earlier configurations prior to the placing of s. Each such configuration is encoded as a *state*, which consists of an array of k numbers $(i_1, i_2, \ldots, i_k)$. The i-th member of the array ϕ is written $\phi[i]$. $\phi[i]$ shows the segment that was last placed in track i. For example, in an instance of three tracks, the state $(2,1,3)$ indicates that s_2 was the last segment placed in track 1, s_1 the last segment in track 2, and s_3 the last segment in track 3. A track with the number 0 implies that no segment has yet been placed on it. For example, $(2,0,3)$.

A new state ϕ is created when a segment is placed. The new state is computed from all the states that was computed when the last segment was placed. More precisely, if we let Φ_m denote the set of states created right after the segment

s_m is placed, then, Φ_m is computed from Φ_{m-1}. The search is complete if at any point, Φ_m is empty, or when the first element in Φ_n is found. Details of this search is shown in Algorithm 2.

Algorithm 2. Decide if there exists solution that fulfill threshold requirement

Input: Segments S, number of tracks k, distance threshold z, radius of curvature r, inter-track distance d.

Output: Whether a placement that fulfills threshold requirement exists.

1: $\Phi_0 \leftarrow \{(0, 0, \ldots, 0)\}$ ▷ $(0, 0, \ldots, 0)$ is an array of length k
2: **for** $m \leftarrow 1, \ldots, n$ **do**
3: $\quad \Phi_m \leftarrow \emptyset$
4: $\quad$ **for** $\phi \in \Phi_{m-1}$ **do**
5: $\quad\quad$ **for** $i \leftarrow 1, \ldots, k$ **do**
6: $\quad\quad\quad$ **if consistent**(s_m, ϕ, i, z) **then** ▷ ϕ allows s_m to be placed on track i
7: $\quad\quad\quad\quad$ Let ϕ' be a copy of ϕ but with the i-th array element set to m.
8: $\quad\quad\quad\quad$ $\Phi_m \leftarrow \Phi_m \cup \{\phi'\}$.
9: **return** true if Φ_n is non-empty, false otherwise.

The function **consistent**(s_m, ϕ, i, z) returns true iff

$$(\forall j, 1 \leq j \leq k)[\ \texttt{dist}(\texttt{ctr}(\phi[j]), \texttt{ctr}(s_m), j, i) \geq z\ ],$$

and that $\phi[i]$ does not overlap s_m. Each call to **consistent**(s_m, ϕ) requires $O(k)$ time to examine.

At each loop of the variable m, there exist at most m^k states in ϕ_{m-1}. Since $m \leq n$, this is bounded by n^k. Hence there are $O(kn^k)$ calls to **consistent**.

On the other hand, the outermost loop of variable n is repeated at most n times. Hence, Algorithm 2 runs in time $O(k^2 n^{k+1})$.

While Algorithm 2 solves the decision problem, it does not return the placement which fulfills the threshold requirement. We now modify Algorithm 2 to return all the possible placements that fulfill the threshold requirement. To find these placements, it suffices that we map each state $\phi \in \Phi_m$ in the earlier algorithm to a set (or array) of (arbitrarily number of) earlier states in Φ_{m-1} from which ϕ can be reached. Each element in `Map` is indexed by ϕ and hence can be dereferenced in constant time. The resultant program, as shown in Algorithm 3, modifies Algorithm 2 only slightly to include the maintenance of the `Map` data structure.

To enumerate all the placements, we call `list_placements`(ϕ, `Map`, ()), as shown below, for each $\phi \in \Phi_n$. The last argument in the call, (), is an empty array.

`list_placements`(ϕ, `Map`, `path`)
1. For $\phi' \in$ `Map`(ϕ)
2. Prepend the track of the last segment placed in ϕ' to `path`.
3. If ϕ' has only one segment placed, output `path`.
4. Otherwise, copy `path` to `path`$'$ and call `list_placements`(ϕ', `Map`, `path`$'$).

Algorithm 3. Find all solutions fulfilling threshold requirement

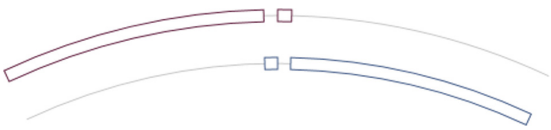
Input: Segments S, number of tracks k, distance threshold z, radius of curvature r, inter-track distance d.

Output: Data structure `Map` for listing all placements f that fulfill the threshold requirement.

$\Phi_0 \leftarrow \{(0, 0, \ldots, 0)\}$ ▷ $(0, 0, \ldots, 0)$ is an array of length k
`Map` $\leftarrow \emptyset$
for $m \leftarrow 1, \ldots, n$ **do**
    $\Phi_m \leftarrow \emptyset$
    for $\phi \in \Phi_{m-1}$ **do**
        for $i \leftarrow 1, \ldots, k$ **do**
            if consistent(s_m, ϕ, i, z) **then** ▷ ϕ allows s_m to be placed on track i
                Let ϕ' be a copy of ϕ but with the i-th array element set to m.
                $\Phi_m \leftarrow \Phi_m \cup \{\phi'\}$.
                Append ϕ to the array `Map`(ϕ'). ▷ Create `Map`(ϕ') if it's undefined.
return `Map` and Φ_n.

It is clear that Algorithm 3 has the same time complexity of $O(k^2 n^{k+1})$ as Algorithm 2. However, `list_placements`'s worst-case runtime is at least k^n, since in the worst case, all k^n placements would fulfill the threshold requirement. Note that at each stage m, each array in `Map` is of length at most m.

Placements obtained using Algorithm 3, however, showed shortcomings in our strategy. Enforcing a constant inter-center distance threshold across all segments ignores differences in the segment lengths. For example, consider the following case where the two longer segments are considered far from both of the two short segments, but the two short segments are considered near each other.



One solution to this is to allow a fixed number of violations to the requirement, which we can decide based on the number of short segments. This mechanism can be achieved by keeping a "violation count" for each state in the array elements of `Map`, through an additional `Violations` array. For instance, suppose `Map`(ϕ) contains the states (ϕ_1, ϕ_2, ϕ_3), then, `Violations`(ϕ) will keep (x_1, x_2, x_3), where x_i is the number of violations in reaching ϕ from ϕ_i. Furthermore, we keep a value `MinViolations`(ϕ) which keeps track of the minimum number of violations among all the states in `Map`(ϕ), that is, min(`Violations`(ϕ)). When computing **consistent**(s_m, ϕ, i, z), `MinViolations`(ϕ) is consulted to decide if the placement is permitted. It is easy to see that this strategy does not add to the time complexity of Algorithm 3. During `list_placements`, the recursion for each state is entered only if the state has not committed more violations than permitted.

An alternative solution that may be more natural is presented in the next section.

5 Maximize Sum of Inter-center Distances

In this section we consider maximizing the sum of inter-center distances between each segment and its nearest segment on the left as well as its nearest segment on the right. More precisely, we define the distance to the nearest segment to the left of a segment s under placement f, $L(s, f)$, as

$$L(s, f) = \min_{s' \in S \setminus \{s\}, \mathtt{ctr}(s') \leq \mathtt{ctr}(s)} \mathtt{dist}(s, s', f(s), f(s')),$$

and similarly we define the distance to the nearest segment to the right of a segment s under placement f as

$$R(s, f) = \min_{s' \in S \setminus \{s\}, \mathtt{ctr}(s') \geq \mathtt{ctr}(s)} \mathtt{dist}(s, s', f(s), f(s')).$$

Our problem is to find placement f which maximizes

$$D = \sum_{s \in S} (L(s, f) + R(s, f))$$

This problem allows a 2-approximation using a polynomial-time dynamic programming which consists of two steps. Each step computes a placement that optimizes $\sum_{s \in S} L(s, f)$ and $\sum_{s \in S} R(s, f)$ respectively. Then, the larger scoring placement among the two is given as the output.

We first consider the placements that optimize $\sum_{s \in S} L(s, f)$. Again, we use a strategy which places segments in the order of $\mathtt{ctr}(s)$. For each of the earlier state ϕ right before the placing of s, we exhaustively examine all the states which extend ϕ by placing s on one of the tracks. For each of these resultant states, we update the sum of inter-center distance with the minimum distance from a segment in ϕ to s. This is shown in Algorithm 4.

The placements can be enumerated using the same recursion given in the earlier section, $\mathtt{list\_placements}(\phi, \mathtt{Map}, ())$, for each $\phi \in \Phi_n$.

To compute the placements which optimize $\sum_{s \in S} R(s, f)$, it suffices that Algorithm 4 is run with the segments sorted in reverse order of $\mathtt{ctr}(s)$.

To see that the method produces a 2-approximation, let D^* denote the optimal D and $\mathbf{D}$ the score Algorithm 4 computed for both steps. Then

$$\begin{aligned} D^* = \max_f \sum_s (L(s, f) + R(s, f)) &\leq \max_f \sum_s L(s, f) + \max_f \sum_s R(s, f) \\ &\leq 2 \max \left(\max_f \sum_s L(s, f), \max_f \sum_s R(s, f) \right) \\ &= 2\mathbf{D}. \end{aligned}$$

Hence $\mathbf{D}$ is of at least $\frac{D^*}{2}$.

Algorithm 4. Find all placements f with maximal $\sum_{s \in S} L(s, f)$

Input: Segments S, number of tracks k, distance threshold z, radius of curvature r, inter-track distance d.

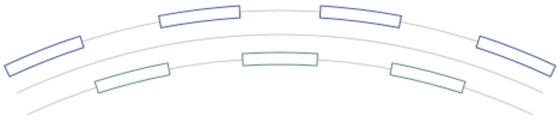
Output: Data structure `Map` for listing all placements f with maximal $\sum_{s \in S} L(s, f)$

$\Phi_0 \leftarrow \{(0, 0, \ldots, 0)\}$ ▷ $(0, 0, \ldots, 0)$ is an array of length k
`Map` $\leftarrow \emptyset$
`SumL`$((0, 0, \ldots, 0)) \leftarrow 0$
for $m \leftarrow 1, \ldots, n$ **do**
    $\Phi_m \leftarrow \emptyset$
    for $\phi \in \Phi_{m-1}$ **do**
        for $i \leftarrow 1, \ldots, k$ **do**
            if consistent(s_m, ϕ, i, z) **then**
                $d \leftarrow$ `SumL`$(\phi) + \min_{1 \le j \le k}$`dist`$(s_{\phi[j]}, s_m, j, i)$
                Let ϕ' be a copy of ϕ but with the i-th array element set to m.
                $\Phi_m \leftarrow \Phi_m \cup \{\phi'\}$.
                if `SumL`(ϕ') is undefined or $d >$ `SumL`(ϕ') **then**
                    `Map`$(\phi') \leftarrow (\phi)$ ▷ New array with ϕ as the first element.
                    `SumL`$(\phi') \leftarrow d$
                else if d is equal to `SumL`(ϕ') **then**
                    Append ϕ to the array `Map`(ϕ').
`Max` $\leftarrow \max\{$`SumL`$(\phi) \mid \phi \in \Phi_n\}$.
$\Phi_n \leftarrow \{\phi \in \Phi_n \mid$ `SumL`(ϕ) is equal to `Max`$\}$
return `Map` and Φ_n.

At this point the computational complexity of the problem is unclear. Our approximation algorithm runs in time $O(k^2 n^{k+1})$, and can be adapted, at the same runtime complexity, to accommodate more contraints from the user, such as requiring specific segments to be placed on given tracks, through modifications to the **consistent** call. The method can also be modified for other classes of distance measures. We do not discuss that in this work.

For the current measure, trimming can be further performed for the algorithm. For instance, for a 4-tracks problem, it is easy to see that a state of $(4, 3, 6, 2)$ is better than a state of $(4, 5, 6, 2)$ if they have the same score. In this case, $(4, 5, 6, 2)$ can be discarded from further investigation.

One problem with maximizing the inter-center distance is that this results in a tendency to place segments on either the top or the bottom tracks since that would maximize their separation. In extreme situations, this may result in placements with empty tracks, such as the following.



In such a situation, we envision that a user would most want to carry out one of these two actions: (1) reduce the number of tracks, or (2) move segments from filled tracks to the empty one. The latter is discussed in the next section.

6 Balanced Placement of Segments on Tracks

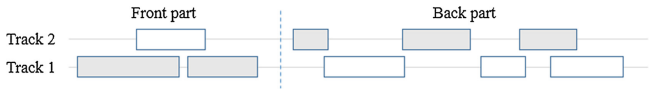
In this section we consider how one can distribute segments evenly over the tracks. Intuitively this problem can be stated as follows: Given segments s_1, s_2, ..., s_n, find placement f that minimizes $\max_{1 \le i,j \le k} |N_i - N_j|$, where N_i is the total length of the segments on track i, that is, $N_i = \sum\{\texttt{ed}(s) - \texttt{st}(s) \mid f(s) = i\}$.

However, such a problem is NP-hard, since it can be used to solve the partition problem as follows. Given an instance of the partition problem of integers $\{a_1, a_2, \ldots, a_n\}$, we create the segments $\{s_1, s_2, \ldots, s_n\}$ where

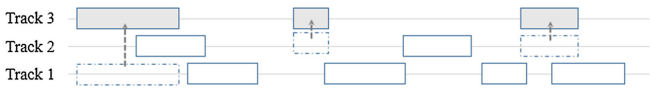
(1) $\texttt{ed}(s_i) = \texttt{st}(s_i) + a_i$, and
(2) $\texttt{st}(s_{i+1}) = \texttt{ed}(s_i) + 1$.

Then, a solution that places the segments evenly onto two tracks solves the partition problem.

In this section we consider a simpler situation, where one track has been left empty in the earlier algorithm, and the task is one of moving segments from other tracks to the empty track. More precisely, given a placement of segments on two tracks, we consider how to move segments from the two tracks to a third track, such that the three tracks now have roughly equal sum of segment lengths. We offer a heuristic solution to this problem in Algorithm 5. For simplity we do not consider the size-changing effects from the curvature of the arcs. Our strategy is to first set a point which divides each track into a front and back part, for instance:



Then, some segments can be selected from the front part of Track 1 and the back part of Track 2 (and vice versa), to form a new track:



We consider all possible points for division, as well as obtaining partitions of various lengths from the divided tracks, as shown in Algorithm 5.

The function **best_partition**(`leftbound`, `rightbound`) is computed as follows.

best_partition(`leftbound`, `rightbound`)
1. Let $R^{\wedge} \leftarrow [1, \texttt{leftbound}]$ and $R^{\$} \leftarrow [\texttt{rightbound}, \texttt{maxangle}]$
2. Let $S_i^x \leftarrow \{s \mid f(s) = i \ \wedge \ \texttt{st}(s), \texttt{ed}(s) \in R^x\}$
3. Let $N_i^x \leftarrow \sum\{\texttt{ed}(s) - \texttt{st}(s) \mid s \in S_i^j\}$
4. Compute for the 2 cases:

   (Case 1) $\texttt{partition}(S_1^{\wedge}, \frac{N_1+N_2}{3} \frac{N_1^{\wedge}}{N_1^{\wedge}+N_2^{\$}})$ and $\texttt{partition}(S_2^{\$}, \frac{N_1+N_2}{3} \frac{N_2^{\$}}{N_1^{\wedge}+N_2^{\$}})$,

(Case 2) $\texttt{partition}(S_2^{\wedge}, \frac{N_1+N_2}{3} \frac{N_2^{\wedge}}{N_2^{\wedge}+N_1^{\$}})$ and $\texttt{partition}(S_1^{\$}, \frac{N_1+N_2}{3} \frac{N_1^{\$}}{N_2^{\wedge}+N_1^{\$}})$.

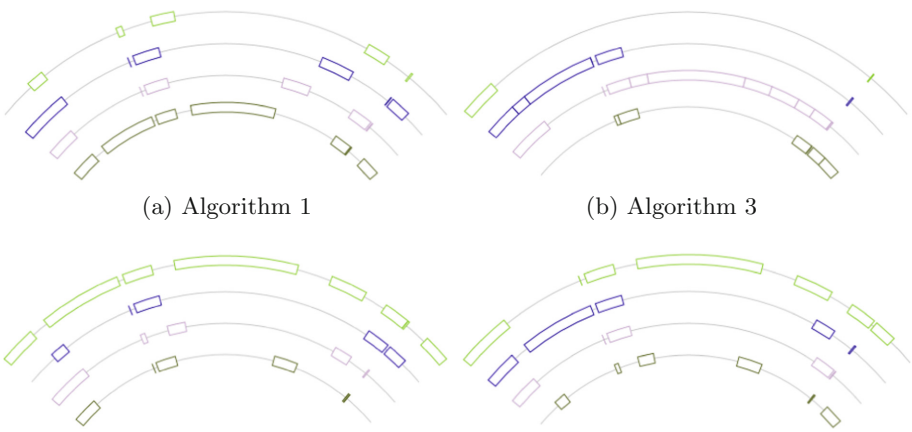
5. If Case 1 results in more even distribution of segment lengths on all three tracks, return the tracks according to the partitioning of Case 1, otherwise return the new tracks formed by the partitioning of Case 2.

Algorithm 5. Heuristic algorithm to evenly distribute segments to new track

Input: Segments S and two-tracks placement f.
Output: Three-tracks placement $\boldsymbol{f}$ which heuristically places S evenly on all tracks.

$\boldsymbol{f} \leftarrow f$
for each track i **do**
  for each segment s in track i **do**
    for each point x in $\{\texttt{st}(s), \texttt{ed}(s)\}$ **do**
      Set `leftbound` and `rightbound` to x, but if there exists some segment s' on track $(3-i)$ overlapping x, then $\texttt{leftbound} \leftarrow \texttt{st}(s')$ and $\texttt{rightbound} \leftarrow \texttt{ed}(s')$
      $f' \leftarrow$ **best_partition**(`leftbound`, `rightbound`)
      Replace $\boldsymbol{f}$ with f' if f' distributes segments more evenly over the tracks.
Output $\boldsymbol{f}$.

The function $\texttt{partition}(S, x)$ accepts a set of segments S and a number x. It computes a partition of S' which sums to x, where $S' = \{\texttt{ed}(s) - \texttt{st}(s) \mid s \in S\}$, and outputs the corresponding partitioning of S. This can be implemented using some approximation algorithm for the partition problem.



(a) Algorithm 1 (b) Algorithm 3

(c) Algorithm 4 (d) 2-approx based on Algorithm 4

Fig. 2. Results from Algorithms 1, 3 and 4

7 Results

In this section we show samples of results from the algorithms proposed earlier, on the same test case of 24 segments.

In Fig. 2(a), we observe that a few segments right after that at the middle has formed a staircase-like pattern, which is consistent with our earlier remarks on Algorithm 1. Nonetheless, the resultant graph is pleasant-looking, and the fast runtime makes the method very usable.

Algorithm 3, on the other hand, has a tendency to not separate segments far apart, due to its lack of a natural mechanism to maximize such separation. The remaining two results from Algorithm 4, which attempt to maximize the sum of inter-center distances, are similarly pleasant-looking, although requiring significantly more time to produce.

References

1. Adey, A., Burton, J.N., Kitzman, J.O., Hiatt, J.B., Lewis, A.P., Martin, B.K., Qiu, R., Lee, C., Shendure, J.: The haplotype-resolved genome and epigenome of the aneuploid hela cancer cell line. Nature **500**(7461), 207–211 (2013)
2. Forment, J.V., Kaidi, A., Jackson, S.P.: Chromothripsis and cancer: causes and consequences of chromosome shattering. Nat. Rev. Cancer **12**(10), 663 (2012)
3. Kataoka, K., Shiraishi, Y., Takeda, Y., Sakata, S., Matsumoto, M., Nagano, S., Maeda, T., Nagata, Y., Kitanaka, A., Mizuno, S., et al.: Aberrant PD-L1 expression through 3′-UTR disruption in multiple cancers. Nature **534**(7607), 402 (2016)
4. Kleinberg, J., Tardos, É.: Algorithm Design. Addison Wesley, Reading (2006)
5. Knuth, D.E.: Art of Computer Programming, Volume 1: Fundamental Algorithms. Addison-Wesley Professional, Reading (1973)
6. Li, S.C., Leong, H.W., Quek, S.K.: New approximation algorithms for some dynamic storage allocation problems. In: Chwa, K.-Y., Munro, J.I.J. (eds.) COCOON 2004. LNCS, vol. 3106, pp. 339–348. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27798-9_37
7. Lipson, D., Capelletti, M., Yelensky, R., Otto, G., Parker, A., Jarosz, M., Curran, J.A., Balasubramanian, S., Bloom, T., Brennan, K.W., et al.: Identification of new ALK and RET gene fusions from colorectal and lung cancer biopsies. Nat. Med. **18**(3), 382 (2012)
8. Nault, J.-C., Datta, S., Imbeaud, S., Franconi, A., Mallet, M., Couchy, G., Letouzé, E., Pilati, C., Verret, B., Blanc, J.-F., et al.: Recurrent AAV2-related insertional mutagenesis in human hepatocellular carcinomas. Nat. Genet. **47**(10), 1187 (2015)
9. Robson, J.M.: Bounds for some functions concerning dynamic storage allocation. J. ACM **21**(3), 491–499 (1974)
10. Soda, M., Choi, Y.L., Enomoto, M., Takada, S., Yamashita, Y., Ishikawa, S., Fujiwara, S., Watanabe, H., Kurashina, K., Hatanaka, H., et al.: Identification of the transforming EML4-ALK fusion gene in non-small-cell lung cancer. Nature **448**(7153), 561 (2007)
11. Stephens, P.J., McBride, D.J., Lin, M.-L., Varela, I., Pleasance, E.D., Simpson, J.T., Stebbings, L.A., Leroy, C., Edkins, S., Mudie, L.J., et al.: Complex landscapes of somatic rearrangement in human breast cancer genomes. Nature **462**(7276), 1005 (2009)

12. Sung, W.-K., Zheng, H., Li, S., Chen, R., Liu, X., Li, Y., Lee, N.P., Lee, W.H., Ariyaratne, P.N., Tennakoon, C., et al.: Genome-wide survey of recurrent HBV integration in hepatocellular carcinoma. Nat. Genet. **44**(7), 765–769 (2012)
13. Waddell, N., Pajic, M., Patch, A.-M., Chang, D.K., Kassahn, K.S., Bailey, P., Johns, A.L., Miller, D., Nones, K., Quek, K., et al.: Whole genomes redefine the mutational landscape of pancreatic cancer. Nature **518**(7540), 495 (2015)
14. Yang, L., Luquette, L.J., Gehlenborg, N., Xi, R., Haseley, P.S., Hsieh, C.-H., Zhang, C., Ren, X., Protopopov, A., Chin, L., et al.: Diverse mechanisms of somatic structural variations in human cancer genomes. Cell **153**(4), 919–929 (2013)

Branch-and-Bound Algorithm for Symmetric Travelling Salesman Problem

Alexey Nikolaev(✉) and Mikhail Batsyn

Laboratory of Algorithms and Technologies for Network Analysis,
National Research University Higher School of Economics,
136 Rodionova street, Nizhny Novgorod, Russia
{ainikolaev,mbatsyn}@hse.ru

Abstract. In this paper a branch-and-bound algorithm for the Symmetric Travelling Salesman Problem (STSP) is presented. The algorithm is based on the 1-tree Lagrangian relaxation. A new branching strategy is suggested in which the algorithm branches on the 1-tree edge belonging to the vertex with maximum degree in the 1-tree and having the maximum tolerance. This strategy is compared with branching on the shortest edge and the so-called strong branching, which is the branching on the edge with maximum tolerance also applied by Held and Karp (1971). The computational experiments show that proposed branching strategy provides better results on TSPlib benchmark instances.

Keywords: Traveling salesman problem · 1-tree
Branch-and-bound algorithm

1 Introduction

The *Traveling Salesman Problem* (TSP) is one of the best known and deeply studied combinatorial optimization problems. It consists of finding the shortest tour for a given set of cities in which every city is visited only once. This problem is equivalent to the problem of finding minimum cost (length) *Hamiltonian circuit* (or *cycle*) for a given graph $G = (V, E)$ with a cost (length) function $c : E \to \mathbb{R}$. If the graph G is an undirected graph, the problem is called *Symmetric Travelling Salesman Problem* (STSP). Otherwise it is called *Asymmetric TSP* (ATSP). A detailed analysis of heuristics and exact algorithms for STSP and ATSP can be found in Laporte (1992) and Matai et al. (2010).

Different relaxations have been proposed for solving the STSP including *2-Matching*, *Subtour Elimination*, *Minimum Spanning Tree* problems. According to the computational results in Reinelt (1994) one of the tightest lower bounds is based on the 1-tree Lagrangian relaxation. This relaxation was originally suggested in Held and Karp (1970). Held and Karp (1971) implemented branch-and-bound algorithm and used the minimum cost 1-tree as the lower bound.

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 311–322, 2018.
https://doi.org/10.1007/978-3-319-94667-2_26

This algorithm was further improved by many researchers (Hansen and Krarup 1974; Volgenant and Jonker 1982; Benchimol et al. 2010). In this paper a branch-and-bound algorithm for the STSP with a new branching strategy is developed. The suggested strategy is compared with the most popular ones.

The paper is organized as follows. The STSP problem and well known approaches for it are given in Sect. 2. Section 3 contains a description of the implemented algorithm. In Sect. 4 computational results are presented and analysed. And conclusions are given in the last section.

2 Problem Description

The integer linear programming model for the STSP problem is presented below.

$$\min \sum_{e \in E} c_e x_e \tag{1}$$

$$\sum_{e \in E | i \in e} x_e = 2 \quad \forall i \in V \tag{2}$$

$$\sum_{e=(i,j) | (i,j) \in E, i,j \in S} x_e \leq |S| - 1 \quad \forall S \subset V,\ 1 < |S| < n \tag{3}$$

$$x_e \in \{0, 1\} \quad \forall e \in E \tag{4}$$

Binary variable x_e is equal to 1 if the edge e is contained in a tour, otherwise it is 0. The objective function (1) minimizes the total cost (length), constraints (2) require that every vertex in the solution has degree 2, and constraints (3) guarantee that there are no disjoint subtours.

The STSP problem can be also formulated as the problem of finding the minimum cost connected subgraph covering all vertices in which every vertex has degree 2. In this interpretation it is obvious that the Minimum Spanning Tree (MST) problem is a relaxation of the STSP in which we do not have any constraints on vertex degrees. MST is the minimum cost connected subgraph over all vertices.

Let us now add two more constraints to this minimum cost subgraph problem. First, we require it to have exactly n edges – the same as the Hamiltonian cycle has and one edge more than the MST has. Second, we choose one of the MST leaves and require it to have degree 2 as every vertex in the Hamiltonian cycle has. We will call this special MST leaf the 1-vertex – v^1.

The 1-vertex v^1 is connected to the MST with its shortest edge and if we add the next shortest edge from v^1, we will obviously obtain the required minimum cost subgraph. This subgraph is called 1-tree and we will call the second shortest 1-vertex edge the 1-edge – e^1.

Thus the minimum cost 1-tree problem is a relaxation of the STSP in which at least one vertex has degree 2. So the solution of this relaxation provides a lower bound to the STSP solution. Moreover, this lower bound can be improved using

the following property, which is actually derived from the Lagrangian relaxation of the STSP (Held and Karp 1971).

Let us choose for every vertex i a certain value π_i and add this value to the cost of each edge incident to this vertex. The cost of each edge (i, j) then becomes equal to $c_{ij}+\pi_i+\pi_j$. In this case an optimal tour for the STSP does not change, and only its length f_{min} increases by $2\sum_{i\in V}\pi_i$. But an optimal 1-tree changes and its cost c^{1tree} in the modified graph provides the lower bound for the increased tour length: $f_{min}+2\sum_{i\in V}\pi_i \geq c^{1tree}$. From it we derive an improved lower bound LB for the original STSP: $f_{min} \geq LB = c^{1tree} - 2\sum_{i\in V}\pi_i$, which depends on the chosen values π_i. And this dependency is a piecewise linear concave function. This means that we can apply a subgradient ascent method to find π_i values providing maximum lower bound.

Since the objective of the STSP is to find the shortest tour, many algorithms apply edge length as a criterion for adding an edge to a solution. The smaller is the edge length, the greater is the probability that this edge belongs to an optimal solution. The first author who suggested a better criterion was Helsgaun (2000). For every edge which does not belong to the minimum cost 1-tree he measures how much we should decrease the edge length, so that this edge appears in the minimum cost 1-tree. Helsgaun (2000) has called this value the α-nearness of an edge to an optimal solution.

In the sensitivity analysis such a value is called the tolerance of an optimization problem parameter. In the same way for an edge belonging to the minimum cost 1-tree its tolerance is measured as the minimal value by which we should increase the edge length, so that it disappears from an optimal 1-tree. An equivalent way to measure this tolerance is to forbid the edge, recompute the minimum cost 1-tree without it, and measure the increase in the 1-tree edges total cost. In the current paper we apply tolerances in our branch-and-bound algorithm.

3 Algorithm

The pseudo-code of the suggested algorithm is provided in Algorithms 1–9. The following parameters and variables are used in the pseudo-code of our algorithm:
$e = (i, j)$ – an edge between vertices i and j: $i = e[1], j = e[2]$
$c(e) = c_e = c_{ij}$ – the cost of edge $e = (i, j)$
$\Delta c(e) = \Delta c_e$ – the tolerance of edge e
deg_i – the degree of vertex i in the current 1-tree
v^* – the vertex with the maximum degree in the 1-tree
deg_{v^*} – the maximum degree of the 1-tree vertices
S^* – the current best TSP solution
f^* – the cost of the current best TSP solution
E – the list of all edges sorted in the non-descending order of edge costs
E^{1tree} – the list of 1-tree edges sorted in the non-ascending order of edge tolerances
v^1 – a special 1-vertex of the 1-tree
e^1_{mst} – the 1-vertex's shortest edge belonging to Minimum Spanning Tree (MST)

e^1 – the next shortest edge from the 1-vertex
c^{1tree} – the total cost of 1-tree edges
E^1 – the list of the 1-vertex edges sorted in the non-descending order of costs
E_a – the list of edges forcibly added to the 1-tree
E_r – the list of edges forcibly removed from the 1-tree and forbidden to be added
$E[j]$ – the j-th element in a sorted list E: $E = (E[1], ..., E[m])$
$\hat{e}$ – an edge which replaces edge e (replacing edge) in the 1-tree if e is forbidden
$\hat{E}^{1tree}$ – the list of the replacing edges for all 1-tree edges
$\hat{e}^1_{mst}$ – the replacing edge for the 1-vertex's shortest edge belonging to MST
$\hat{e}^1$ – the replacing edge for the next shortest edge from the 1-vertex
$1tree$ – the 1-tree including: $\{E^{1tree}, v^1, e^1_{mst}, e^1, \hat{E}^{1tree}, \hat{e}^1_{mst}, \hat{e}^1, \Delta c, deg, v^*\}$
V_l, V_r – vertex sets of left and right MST subtrees separated by a removed edge
$k \leftarrow$ GetIndex(E, e) – returns index k of element e in a sorted list E: $e = E[k]$
s – the name of the selected branching strategy

The main procedure is presented in Algorithm 1. First we find initial solution by applying the LKH heuristic (Helsgaun 2000) with default parameters. Then following the approach of Held and Karp (1970, 1971) we find the values of vertex weights π_i added to edge costs in order to improve the 1-tree lower bound. We run the SubgradientAscentForBest1Tree() function only once and the obtained edge costs are used then in all nodes of the BranchAndBound() algorithm. This is the main difference between our algorithm and other ones (Held and Karp 1971; Hansen and Krarup 1974) which use subgradient ascent for all branch-and-bound subproblems.

Algorithm 1. Main branch-and-bound algorithm

```
function BranchAndBoundAlgorithm()
    s ← Choose Branching Strategy
    S*, f* ← LKH()
    1tree, [π_i] ← SubgradientAscentForBest1Tree()
    Π ← 2 Σ_{i∈V} π_i,  [c_ij] ← [c_ij] + [π_i] + [π_j],  E_a ← ∅,  E_r ← ∅
    BranchAndBound(1tree, ∅, ∅)
    return S*, f*
```

The subgradient ascent method to obtain a better 1-tree bound is provided in Algorithm 2. This algorithm is the same as the **1tree_bound procedure** from Reinelt (1994) and all the values of its parameters are taken from the book. At each iteration we construct a minimum cost 1-tree in Build1Tree() function (Algorithm 3), compute the subgradient direction $[d_i]$ using 1-tree vertex degrees $[deg_i]$, and recompute the current vertex weights moving partly in this direction $[d_i]$ and partly in the previous step direction $[d^0_i]$ by the current step size α. The 1-tree giving the best lower bound is stored and returned in $1tree^*$ object.

After all iterations we run ComputeDegreesAndTolerances() function (Algorithm 4) to compute the degrees of vertices $[deg_i]$, determine the vertex v^* with maximum degree in the 1-tree, compute tolerances and replacing edges in ComputeTolerance() function, sort edges by tolerances, and prepare the list E^1 of 1-vertex edges sorted by costs.

Algorithm 2

function SUBGRADIENTASCENTFORBEST1TREE()
  $1tree \leftarrow$ BUILD1TREE()
  $LB^* \leftarrow c^{1tree},\ \alpha \leftarrow 10 \cdot (f^* - LB^*)/|V|$
  $\lambda \leftarrow 0.98,\ [\pi_i] \leftarrow 0,\ [c_{ij}^0] \leftarrow [c_{ij}],\ [d_i^0] \leftarrow 0$
  for $(k \leftarrow 1;\ k \leq 300;\ k \leftarrow k+1)$ **do**
    $[d_i] \leftarrow [deg_i] - 2$
    $[\pi_i] \leftarrow [\pi_i] + \alpha \cdot (0.7d_i + 0.3d_i^0)$
    $[c_{ij}] \leftarrow [c_{ij}^0] + [\pi_i] + [\pi_j]$
    $1tree \leftarrow$ BUILD1TREE()
    $\Pi \leftarrow 2\sum_{i \in V} \pi_i,\ LB \leftarrow c^{1tree} - \Pi$
    if $(LB > LB^*)$ **then**
      $[\pi_i^*] \leftarrow [\pi_i],\ LB^* \leftarrow LB,\ 1tree^* \leftarrow 1tree$
    $[d_i^0] \leftarrow [d_i],\ \alpha \leftarrow \lambda\alpha$
  $[c_{ij}] \leftarrow [c_{ij}^0]$
  $1tree^* \leftarrow$ COMPUTEDEGREESANDTOLERANCES($1tree^*$)
  return $[\pi_i^*],\ 1tree^*$

Construction of a minimum cost 1-tree is described in Algorithm 3. First we build the Minimum Spanning Tree (MST) applying the well-known Kruskal approach (KRUSKALMSTALGORITHM()). To obtain a 1-tree we then choose one of the MST leaves as a special 1-vertex v^1 of our 1-tree and connect this vertex with the shortest possible edge $e^1 = (i, j^*)$, called 1-edge, to the MST. We also compute the next shortest edge $(i, \hat{j}^*)$ as the replacing edge for the 1-edge to compute the 1-edge tolerance later. Since every vertex can be assigned to be the 1-vertex, we use the vertex for which the 1-edge is longer, because it increases our lower bound. Next we add the 1-edge to our tree and compute its tolerance.

Algorithm 3

function BUILD1TREE()
  $E^{1tree} \leftarrow$ KRUSKALMSTALGORITHM(V, E)
  $e^1 \leftarrow \emptyset,\ c^1 \leftarrow 0$
  for $(i \in V)$ **do** ▷ choose the 1-vertex from MST leaves
    if $(deg_i \neq 1)$ **continue**
    $E_i \leftarrow \{e \in E^{1tree} \mid i \in e\}$
    $e \leftarrow E_i[1]$ ▷ there is only one element since $deg_i = 1$
    $c^* \leftarrow \infty,\ j^* \leftarrow 0$
    for $(j \in V\ \&\ j \notin e)$ **do** ▷ find the 1-edge – 2nd shortest edge from 1-vertex
      if $(c_{ij} < c^*)$ **then**
        $c^* \leftarrow c_{ij},\ \hat{j}^* \leftarrow j^*,\ j^* \leftarrow j$
    if $(c^* > c^1)$ **then** ▷ we choose the 1-vertex with the longest 1-edge
      $c^1 \leftarrow c^*,\ e^1 \leftarrow (i, j^*),\ \hat{e}^1 \leftarrow (i, \hat{j}^*),\ e^1_{mst} \leftarrow e,\ \hat{e}^1_{mst} \leftarrow e^1,\ v^1 \leftarrow i$
  $E^{1tree} \leftarrow E^{1tree} \cup e^1,\ \hat{E}^{1tree} \leftarrow \{\hat{e}^1\},\ \Delta c(e^1) = c(\hat{e}^1) - c(e^1)$
  $1tree \leftarrow \{E^{1tree}, v^1, E^1, e^1_{mst}, e^1, \hat{E}^{1tree}, \hat{e}^1_{mst}, \hat{e}^1, \Delta c, [deg_i], v^*\}$
  return $1tree$

To illustrate our algorithm hereafter we will use the graph shown in Fig. 1a. The 1-tree for this graph is shown in Fig. 1b. It is constructed from the MST by adding 1-edge $e^1 = (5, 1)$. The MST is built with Kruskal algorithm by sequentially choosing the shortest edges: $(2, 4), (3, 5), (3, 1)$, and $(1, 4)$. The MST has two leaves 2 and 5, and we choose leaf 5 as the 1-vertex v^1 because its second shortest edge has length 7, which is greater than length 6 of the second shortest edge of leaf 2.

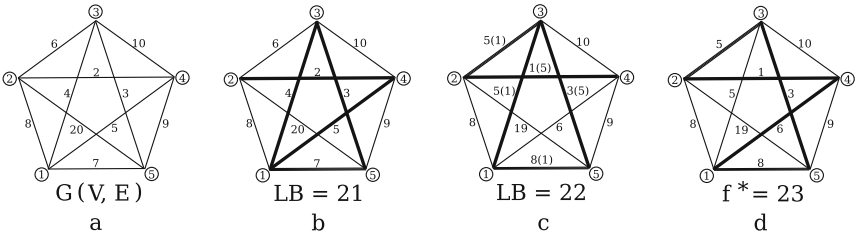


Fig. 1. Example for algorithm illustration

We also show one iteration of the subgradient ascent method in Fig. 1c. To have better numbers we use the initial step value $\alpha = 10/7$, so that after the first iteration we have for vertex 1 with degree 3: $\pi_1 = 10/7 \cdot (0.7 \cdot 1 + 0.3 \cdot 0) = 1$ and for vertex 2 with degree 1: $\pi_2 = 10/7 \cdot (0.7 \cdot (-1) + 0.3 \cdot 0) = -1$. This gives the 1-tree shown in Fig. 1c.

The tolerances of the 1-tree edges are shown in brackets in Fig. 1c. Their computation is performed by ComputeTolerance() function (Algorithm 9) which calls FindReplacingEdge() function (Algorithm 8) to determine the replacing edge. The tolerance of a 1-tree edge is computed as the value to which the 1-tree cost increases if we forbid this edge and have to replace it with another longer edge called replacing edge in this paper. When we forbid a 1-tree edge adding it to the list of removed edges E_r, the 1-tree breaks into a left subtree (with respect to the removed edge) with vertex set V_l and a right subtree with vertex set V_r. We represent a 1-tree as a tree data structure and so it is easy to obtain these vertex sets V_l and V_r applying a standard recursive traverse procedure – TraverseTree() function in our pseudo-code. Since all edges in list E are sorted by Kruskal algorithm in the non-descending order of their costs, we return as the replacing edge the first available edge from E which connects the two subtrees. For example edge $(2, 4)$ in Fig. 1c has the replacing edge $(1, 4)$ and so its tolerance is equal to 5.

The recursive branch-and-bound procedure is presented in Algorithm 5. The first time this procedure is called without any forcibly added or removed edge: $e_a = e_r = \emptyset$. In this case it actually skips the first if-else blocks and proceeds to choosing the edge according to the chosen branching strategy (function BranchingStrategy()). On the left branch this edge is forcibly added to the 1-tree, and on the right branch it is forcibly removed from the 1-tree and forbidden. For both branches the branch-and-bound procedure is then recursively called.

Algorithm 4

function ComputeDegreesAndTolerances($1tree$)
  $[deg_i] \leftarrow |\{e \in E^{1tree} \mid i \in e\}|$, $v^* \leftarrow \text{argmax}_{i \in V}\,(deg_i)$
  for $(e \in E^{1tree})$ **do**
    ComputeTolerance($E^{1tree}, \hat{E}^{1tree}, \emptyset, e$)
  Sort($E^{1tree}, \hat{E}^{1tree}$) ▷ sort in non-ascending order of tolerances
  for $(j \in V \setminus v^1)$ **do**
    $E^1 \leftarrow E^1 \cup (v^1, j)$ ▷ insert keeping sorting in non-descending order of costs
  return $1tree$

On the left branch where we have forcibly added edge e_a we check for each vertex of this edge if there is one more edge from this vertex forcibly added to the 1-tree. In this case such a vertex already has two guaranteed edges in the 1-tree and so all other edges from this vertex should be forbidden, because every vertex should have degree 2 in the final feasible TSP solution. We add such forbidden edges to list E'_r and then also to global list E_r. If such an edge belongs to the current 1-tree we replace it calling the ReplaceEdge() function. It is not possible to replace a forbidden edge, we restore the global list of removed edges and return from recursion, since there are no feasible solutions on this branch.

On the right branch where we have removed edge e_r we first compute the lower bound LB using the edge tolerance value. And if we already have a solution f^* not worse than this bound, then we prune this branch. Otherwise we replace the removed edge calling the ReplaceEdge() function.

On both branches if some edges have been replaced in the 1-tree we have to recompute all edge tolerances in RecomputeTolerances() function (Algorithm 9). If the maximum vertex degree deg_{v^*} in the 1-tree is equal to 2, then our 1-tree is a feasible TSP solution – a cycle, because of the following proposition.

Proposition 1. *If the maximum degree of 1-tree vertices is 2, then this 1-tree is a cycle.*

Proof. A 1-tree with n vertices has exactly n edges. So the sum of vertex degrees is equal to $2n$. Since the maximum degree is 2, then the sum of n vertex degrees can be equal to $2n$ only if every vertex degree is equal to 2. Otherwise, this sum will be less than $2n$. This means that this 1-tree is a cycle, because every 1-tree is a connected graph. □

For our example in Fig. 1c the vertex with maximum degree is $v^* = 3$ and its edge with maximum tolerance is $e^* = (3, 5)$. First we go to the left branch setting $e_a = (3, 5)$ and $E_a = \{(3, 5)\}$. This does not change the optimal 1-tree and we branch again on the edge from vertex $v^* = 3$ having next maximum tolerance – edge $(3, 2)$. Here on the left branch we have two forcibly added edges from vertex 3: $E_a = \{(3, 5), (3, 2)\}$. So we forbid all its other edges: $E_r = E'_r = \{(3, 1), (3, 4)\}$. Edge $(3, 1)$ belonging to the 1-tree is replaced with edge $(4, 1)$ and we get the 1-tree shown in Fig. 1d. Its maximum degree is 2 and so it is our first feasible solution with $f^* = 23$. On the right branch we forbid edge $(3, 2)$: $E_a = \{(3, 5)\}, E_r = \{(3, 2)\}$. The tolerance of this edge immediately gives us the

Algorithm 5. Recursive branch-and-bound procedure

function BranchAndBound($1tree, e_a, e_r$)
  $E'_r \leftarrow \emptyset,\ \hat{e}_r \leftarrow \emptyset$
  if $(e_a \neq \emptyset)$ **then**
    for $(i \in e_a)$ **do** ▷ repeat for both vertices of this edge: $i = e_a[1], i = e_a[2]$
      $E_i \leftarrow \{e \in E_a \mid i \in e\}$ ▷ a list of forcibly added edges from vertex i
      if $(|E_i| = 2)$ **then** ▷ if there are already two edges from vertex i
        for $(j \in V)$ **do** ▷ other edges from vertex i will be forbidden
          if $((i,j) \notin E_i)\ E'_r \leftarrow E'_r \cup (i,j)$
    for $(e' \in E'_r)$ **do**
      $E_r \leftarrow E_r \cup e'$
      if $(e' \in E^{1tree})$ **then**
        $\hat{e}' \leftarrow$ ReplaceEdge($1tree, E_r, e$)
        if $(\hat{e}' = \emptyset)$ **then**
          $E_r \leftarrow E_r \setminus E'_r$ ▷ restore the list of forbidden edges
          return
  else if $(e_r \neq \emptyset)$ **then**
    $LB \leftarrow c^{1tree} - \Pi + \Delta c(e_r)$
    if $(LB \geq f^*)$ **return**
    $\hat{e}_r \leftarrow$ ReplaceEdge($1tree, E_r, e_r$)
    if $(\hat{e}_r = \emptyset)$ **return**
  if $(deg_{v^*} = 2\ \&\ c^{1tree} - \Pi < f^*)$ **then** ▷ a feasible and better TSP solution
    $f^* \leftarrow c^{1tree} - \Pi,\ S^* \leftarrow E^{1tree},\ E_r \leftarrow E_r \setminus E'_r$
    return
  if $(E'_r \neq \emptyset\ or\ \hat{e}_r \neq \emptyset)$ RecomputeTolerances($E^{1tree}, \hat{E}^{1tree}, E_r$)
  $e^* \leftarrow$ BranchingStrategy($1tree$)
  $E_a \leftarrow E_a \cup e^*$ ▷ forcibly add edge e^*
  BranchAndBound($1tree, e^*, \emptyset$)
  $E_a \leftarrow E_a \setminus e^*,\ E_r \leftarrow E_r \cup e^*$ ▷ forbid edge e^*
  BranchAndBound($1tree, \emptyset, e^*$)
  $E_r \leftarrow E_r \setminus e^*,\ E_r \leftarrow E_r \setminus E'_r$ ▷ restore the list of forbidden edges
function BranchingStrategy($1tree$)
  if $s =$ "max degree" **then** ▷ choose from the maximum-degree vertex edges
    $E^* \leftarrow \{e \in E^{1tree} \setminus E_a \mid v^* \in e\}$ ▷ $E^*[1]$ has maximum tolerance
  else if $s =$ "max tolerance" **then** ▷ choose from 1-tree edges
    $E^* \leftarrow E^{1tree} \setminus E_a$ ▷ $E^*[1]$ has maximum tolerance
  else if $s =$ "min length" **then** ▷ choose from 1-tree edges sorted by length
    $E^* \leftarrow E \cap E^{1tree} \setminus E_a$ ▷ $E^*[1]$ has minimum length due to list E sorting
  return $E^*[1]$

lower bound $LB = 22 + 1 = 23$. So we prune this branch and return back to our first branching. Now we forbid edge $(3,5)$: $E_a = \emptyset, E_r = \{(3,5)\}$, and get the lower bound $LB = 22 + 5 = 27$, which again allows us to prune the branch.

Function ReplaceEdge() is presented in Algorithm 6. It has a special processing for 1-vertex edges e^1_{mst}, the shortest edge from 1-vertex, and e^1, the

Algorithm 6. Replaces a forbidden edge in 1-tree with a new edge

function ReplaceEdge($1tree, E_r, e_r$)
  $E_r \leftarrow E_r \cup e_r$
  $E^{1tree} \leftarrow E^{1tree} \setminus e_r$
  $c^{1tree} \leftarrow c^{1tree} - c(e_r)$
  $\hat{E}^{1tree} \leftarrow \hat{E}^{1tree} \setminus \hat{e}_r$
  if ($e_r = e^1_{mst}$ or $e_r = e^1$) **then**
    if ($\hat{e}^1 = \emptyset$) **return** $\emptyset$
    $k \leftarrow$ GetIndex($E^1, \hat{e}^1$)
    if ($e_r = e^1_{mst}$) $e^1_{mst} \leftarrow e^1$
    $e^1 \leftarrow \hat{e}^1,\ \hat{e}^1_{mst} \leftarrow \hat{e}^1,\ e \leftarrow \hat{e}^1,\ \hat{e}^1 \leftarrow \emptyset$
    $\Delta c(e^1_{mst}) \leftarrow c(\hat{e}^1_{mst}) - c(e^1_{mst})$
    for ($k \leftarrow k+1; k \le |E^1|; k \leftarrow k+1$) **do**
      if ($E^1[k] \notin E_r$) **then**
        $\hat{e}^1 \leftarrow E^1[k]$
        break
    $\hat{e} \leftarrow \hat{e}^1$
  else
    if ($\hat{e}_r = \emptyset$) **return** $\emptyset$
    $e \leftarrow \hat{e}_r$
    $\hat{e} \leftarrow$ FindReplacingEdge(E^{1tree}, E_r, e)
  $\Delta c(e) \leftarrow c(\hat{e}) - c(e)$
  $E^{1tree} \leftarrow E^{1tree} \cup e$ ▷ insert e keeping the sorting by tolerances
  $c^{1tree} \leftarrow c^{1tree} + c(e)$
  $\hat{E}^{1tree} \leftarrow \hat{E}^{1tree} \cup \hat{e}$
  RecomputeDegrees($[deg_i], v^*, e_r, e$)

Algorithm 7. Recomputes degrees of the 1-tree vertices

function RecomputeDegrees($[deg_i], v^*, e_r, e$)
  for ($i \in e_r$) **do**
    $deg_i \leftarrow deg_i - 1$
  for ($i \in e$) **do**
    $deg_i \leftarrow deg_i + 1$
  $v^* \leftarrow \text{argmax}_{i \in V}\,(deg_i)$ ▷ we use a sorted array $[deg_i]$ to get v^*

Algorithm 8. Finds the replacing edge for the given edge in the 1-tree

function FindReplacingEdge(E^{1tree}, E_r, e)
  $i \leftarrow e[1],\ j \leftarrow e[2]$
  $V_l \leftarrow$ TraverseTree(E^{1tree}, i)
  $V_r \leftarrow$ TraverseTree(E^{1tree}, j)
  for ($\hat{e} \in E \setminus E_r$) **do** ▷ E is sorted in the non-descending order of edge costs
    if ($\hat{e} \cap V_l \neq \emptyset$ & $\hat{e} \cap V_r \neq \emptyset$) **then**
      return $\hat{e}$
  return $\emptyset$

next shortest edge from 1-vertex. And for other MST edges it calls FindReplacingEdge() (Algorithm 8) which has been described above. For efficient finding of the replacing edge for one of the 1-vertex edges we use the list E^1 of 1-vertex edges sorted in the non-descending order of their lengths. The replacing edge for edge e^1_{mst} is the 1-edge e^1, and for the 1-edge – the next shortest edge $\hat{e}^1$. After we have replaced the edges of the 1-vertex we update the replacing edge $\hat{e}^1$ with the next available shortest edge from list E^1. We also decrease the degrees of the removed edge vertices, increase the degrees of the inserted edge vertices, and update the maximum degree vertex v^* in RecomputeDegrees() function (Algorithm 7).

Algorithm 9. Computes tolerances for 1-tree edges

function RecomputeTolerances($E^{1tree}, \hat{E}^{1tree}, E_r$)
  for ($e \in E^{1tree}$) **do**
    ComputeTolerance($E^{1tree}, \hat{E}^{1tree}, E_r, e$)
function ComputeTolerance($E^{1tree}, \hat{E}^{1tree}, E_r, e$)
  $E_r \leftarrow E_r \cup e$
  $\hat{E}^{1tree} \leftarrow \hat{E}^{1tree} \setminus \hat{e}$
  $\hat{e} \leftarrow$ FindReplacingEdge(E^{1tree}, E_r, e)
  $\hat{E}^{1tree} \leftarrow \hat{E}^{1tree} \cup \hat{e}$
  $\Delta c(e) \leftarrow c(\hat{e}) - c(e)$
  $E_r \leftarrow E_r \setminus e$

4 Computational Results

The branch-and-bound algorithm is implemented in C\C++ and tested on TSPlib (Reinelt 1991) instances with the number of cities not greater than 130. All experiments are carried out on a computer with 2.2 GHz CPU and 8 GB of RAM. On each test instance each algorithm is executed with 300 s time limit.

Table 1 presents the computational results for the considered TSPlib instances. The table contains information about the performance and the search tree size of the branch-and-bound algorithm with different branching strategies. The value "–" means that the algorithm exceeds the time limit. Three different strategies have been implemented:

- **Strategy 1 "max tolerance"**. The edge with maximum tolerance is chosen from the 1-tree. This strategy is suggested by Held and Karp (1971).
- **Strategy 2 "min length"**. The shortest edge is selected from the 1-tree.
- **Strategy 3 "max degree"**. The vertex with maximum degree is found and the edge with the maximum tolerance incident to this vertex is chosen.

According to the computational results the suggested "max degree" strategy is the best with respect to the search tree size and the computational time. "Max tolerance" strategy has the worst performance. A potential reason for it is that the computation of all tolerances is computationally expensive. It implies that strategy 2 and 3 are more preferable in practice.

Table 1. Comparison of different branching strategies

	Strategy 1 "max tolerance"		Strategy 2 "min length"		Strategy 3 "max degree"	
Instance	Number of steps	Time, s	Number of steps	Time, s	Number of steps	Time, s
att48	123	**0.01**	125	**0.01**	**60**	**0.01**
bayg29	29	**0**	30	**0**	**21**	**0**
bays29	41	**0**	59	**0**	**22**	**0**
berlin52	**1**	**0**	**1**	**0**	**1**	**0**
brazil58	323	0.04	1455	0.06	**83**	**0.01**
burma14	**1**	**0**	**1**	**0**	**1**	**0**
dantzig42	87	**0.01**	130	**0.01**	**46**	**0.01**
eil51	**2012**	**0.35**	32215	1.44	17053	0.77
eil76	**650**	0.23	1775	**0.19**	3617	0.42
eil101	**1727**	1.38	3418	**0.6**	21497	4.62
fri26	**1**	**0**	**1**	**0**	**1**	**0**
gr17	**1**	**0**	**1**	**0**	**1**	**0**
gr21	**1**	**0**	**1**	**0**	**1**	**0**
gr24	29	**0**	18	**0**	**10**	**0**
gr48	12931	1.98	**8656**	**0.38**	19194	0.98
gr96	-	-	-	-	**444724**	**66.71**
hk48	60	0.02	70	**0.01**	**30**	**0.01**
kroB100	-	-	**245919**	**35.73**	-	-
kroC100	-	-	1943807	282.37	**375393**	**67.02**
kroD100	81324	51.16	38952	5.3	**28638**	**5.09**
lin105	108	0.09	95	0.02	**20**	**0.01**
rat99	6605	1.9	3136	0.39	**1418**	**0.22**
rd100	506	0.26	1459	0.15	**244**	**0.04**
st70	**7206**	2.63	46547	3.37	26681	**2.53**
swiss42	48	0.01	81	**0**	**26**	**0**
ulysses16	**1**	**0**	**1**	**0**	**1**	**0**
ulysses22	**1**	**0**	**1**	**0**	**1**	**0**

5 Conclusion

In this paper we developed a branch-and-bound algorithm based on the 1-tree Lagrangian relaxation with a new branching strategy and considered other branching strategies for solving the STSP. We compared our strategy with the so-called strong branching applied by Held and Karp (1971) and with branching by the shortest edge. The computational results show that the developed branching strategy provides better results.

Acknowledgments. The research was funded by Russian Science Foundation (RSF project No. 17-71-10107).

References

Benchimol, P., Régin, J.C., Rousseau, L.M., Rueher, M., van Hoeve, W.J.: Improving the Held and Karp approach with constraint programming. In: Lodi, A., Milano, M., Toth, P. (eds.) CPAIOR 2010. LNCS, vol. 6140, pp. 40–44. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13520-0_6

Hansen, K.H., Krarup, J.: Improvements of the Held-Karp algorithm for the symmetric traveling-salesman problem. Math. Program. **7**(1), 87–96 (1974)

Held, M., Karp, R.M.: The traveling-salesman problem and minimum spanning trees. Oper. Res. **18**(6), 1138–1162 (1970)

Held, M., Karp, R.M.: The traveling-salesman problem and minimum spanning trees: Part II. Math. Program. **1**(1), 6–25 (1971)

Helsgaun, K.: An effective implementation of the Lin-Kernighan traveling salesman heuristic. Eur. J. Oper. Res. **126**(1), 106–130 (2000)

Laporte, G.: The traveling salesman problem: an overview of exact and approximate algorithms. Eur. J. Oper. Res. **59**(2), 231–247 (1992)

Matai, R., Singh, S., Mittal, M.L.: Traveling salesman problem: an overview of applications, formulations, and solution approaches. In: Traveling Salesman Problem, Theory and Applications. InTech (2010)

Reinelt, G.: TSPLIB-A traveling salesman problem library. ORSA J. Comput. **3**(4), 376–384 (1991)

Reinelt, G.: The Traveling Salesman: Computational Solutions for TSP Applications. Lecture Notes in Computer Science. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48661-5

Volgenant, T., Jonker, R.: A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation. Eur. J. Oper. Res. **9**(1), 83–89 (1982)

LZ-ABT: A Practical Algorithm for α-Balanced Grammar Compression

Tatsuya Ohno[1], Keisuke Goto[2], Yoshimasa Takabatake[1], Tomohiro I[1], and Hiroshi Sakamoto[1(✉)]

[1] Kyushu Institute of Technology, Kitakyushu, Japan
t_ohno@donald.ai.kyutech.ac.jp,
{takabatake,tomohiro,hiroshi}@ai.kyutech.ac.jp
[2] Fujitsu Laboratories Ltd., Kawasaki, Japan
goto.keisuke@jp.fujitsu.com

Abstract. We propose a new LZ78-style grammar compression algorithm, named *LZ-ABT*, which is a simple online algorithm to create, given a string of length N over an alphabet of size σ, an α-balanced grammar in $O(N \log N \log \sigma)$ time and $O(n)$ space in addition to the input string, where n is the grammar size to output. LZ-ABT can avoid the lower-bound of $\Omega(N^{5/4})$ time of the naive algorithms for LZMW and LZD, other LZ78-style compression algorithms, which was observed in [Badkobeh et al. SPIRE 2017, pp. 51–67]. We also show that the algorithm can be executed in compressed space, i.e., without storing the whole input string explicitly in memory: in $O(N \log^2 N \log \sigma)$ time and $O(n)$ space, or $O(N \log N \log \sigma)$ time and $O(n \log^* N)$ space. We implement LZ-ABT running in $O(N \log N \log \sigma)$ time and $O(N)$ space and empirically show that its performance is competitive to LZD. This is the first practical implementation of α-balanced grammar compression to the best of our knowledge.

1 Introduction

Grammar compression is a model of lossless compression in which a compressed string is represented by a context-free grammar that deterministically derives the string only. In the last two decades, grammar compression has been extensively studied not only because it is theoretically clean enough to work on [9], but also because it can actually model many practical compression algorithms (like LZ78 [17], Bisection [11], SEQUITUR [12], RePair [8], LZD [4], LZMW [10], and SOLCA [16]). While the problem of computing the smallest grammar for an input string is known to be NP-hard [2,15], several polylogarithmic approximation algorithms have been proposed [2,6,7,13,14,16]. Interestingly, almost all these algorithms generate a *balanced* grammar in the sense that the height of the derivation tree of any variable is the logarithmic order of the length of the string it derives. As is usual with balanced trees, it is certain that the balancedness is helpful for improving the worst-case behavior in grammar compression algorithms. In this paper, we focus on α-balanced grammars, which were employed in

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 323–335, 2018.
https://doi.org/10.1007/978-3-319-94667-2_27

the seminal paper [2] in this field but have not been well-studied up until today. We apply the α-balanced property to LZD [4], a practical LZ78-style grammar compression, to improve its worst-case running time for anomalous input strings.

LZD [4] was recently proposed to improve the compression ability of LZ78. Basically, LZD greedily parses a string into a sequence of phrases from left to right while searching for the longest prefix of the unprocessed string that matches a phrase created so far. A naive implementation of the longest-match string searching is to conduct a prefix search on the Patricia tree that stores the phrases. It has been shown by experimental results that this naive implementation is simple enough to run fast in practical. If the number of character-to-character comparisons in a single prefix search is within a constant multiple of the obtained length of the longest prefix, the algorithm would actually run in (almost) linear time. However, in theory the prefix search could badly "overrun", and it was reported in [1] that the algorithm has a lower-bound of $\Omega(N^{5/4})$ time.

In this paper, we propose a new LZ78-style grammar compression algorithm, named *LZ-ABT*, which is a simple online algorithm to create, given a string of length N over an alphabet of size σ, an α-balanced grammar in $O(N \log N \log \sigma)$ time and $O(n)$ space in addition to the input string, where n is the grammar size to output.[1] In contrast to LZD, we impose the α-balanced property to the grammar rules, i.e., for any derivation rule $X \to X_\ell X_r$, the fraction of two strings derived by X_ℓ and X_r is bounded by a constant with respect to predefined $0 < \alpha \leq (3-\sqrt{5})/2$. Since this property suppresses the bad "overrun" of LZD, we can evade the lower-bound of $\Omega(N^{5/4})$ time. In addition, the α-balanced property enables us to execute the algorithm efficiently in compressed space without storing the whole input string explicitly in memory: in $O(N \log^2 N \log \sigma)$ time and $O(n)$ space, or $O(N \log N \log \sigma)$ time and $O(n \log^* N)$ space. The latter utilizes bookmarking data structures on balanced grammars, which can be maintained in an online manner.

We implement LZ-ABT running in $O(N \log N \log \sigma)$ time and $O(N)$ space and empirically show that its performance is competitive to LZD [4]. We also observe LZ-ABT keeps its performance in malicious examples for other LZ78-style compression algorithms [1]. This is the first practical implementation of α-balanced grammar compression to the best of our knowledge.

2 Preliminaries

2.1 Notation

Let Σ be an ordered *alphabet*. An element of Σ^* is called a *string*. The length of a string S is denoted by $|S|$. The empty string ε is the string of length 0, namely, $|\varepsilon| = 0$. For a string $S = xyz$, strings x, y, and z are called a *prefix*, *substring*, and *suffix* of S, respectively. For $1 \leq i \leq |S|$, the ith character of a

[1] We remark that the $\log \sigma$ multiplicative factor in the running time is the cost to conduct a binary search at internal nodes in the Patricia tree, and can be removed by using hash function if we allow its non-deterministic behavior.

string S is denoted by $S[i]$. For $1 \leq i \leq j \leq |S|$, let $S[i..j] = S[i] \cdots S[j]$, i.e., $S[i..j]$ is the substring of S starting at position i and ending at position j in S. For convenience, let $S[i..j] = \varepsilon$ if $j < i$.

The default base of log we use is two unless otherwise noted. Let $\log^* m$ denote the iterated logarithm of m. Let $\uparrow\uparrow$ denote the up-arrow notation of Knuth, in particular, $2 \uparrow\uparrow 0 = 1$ and $2 \uparrow\uparrow (k+1) = 2^{2\uparrow\uparrow k}$ for any $k \geq 0$. For any $m \geq 0$, we use $\tau(m)$ to denote the set $\{2 \uparrow\uparrow k \mid k \geq 1, 2 \uparrow\uparrow k \leq m\}$. Clearly, $|\tau(m)| = O(\log^* m)$.

2.2 α-balanced Grammars

For a fixed constant $0 < \alpha < 1/2$, two positive integers ℓ and r are said to be α-balanced iff the following condition holds, where $m = \ell + r$ and $\beta = (1-\alpha)/\alpha$:

$$\begin{aligned} & \ell \geq \alpha m \text{ and } r \geq \alpha m \\ \Leftrightarrow\ & (1-\alpha)m \geq \ell \text{ and } (1-\alpha)m \geq r \\ \Leftrightarrow\ & r/\ell \leq \beta \text{ and } \ell/r \leq \beta. \end{aligned}$$

The larger α imposes a stronger balancing condition on the two integers. Extending this notion, two strings are called α-balanced iff their lengths are α-balanced.

Grammar compression is a model of lossless compression in which we consider a context-free grammar that deterministically derives a single string as a compressed representation of the string. In particular, we focus on a normal form such that the righthand side of every rule is of size two[2], namely, its derivation tree forms a full binary tree: an internal node of the derivation tree is labeled with a variable of the grammar, and a leaf is labeled with a terminal (character). Throughout this paper, we denote the set of variables by V and the set of rules by D, and assume that, given $X \in V$, we can access the rule $X \to X_\ell X_r \in D$ in constant time. Let $val(X)$ denote the string X derives. The size $|X|$ of X is $|val(X)|$, and the height of X is the height of the derivation tree of X. The height of a grammar is defined by the height of the starting symbol. Let $[V] = \{val(X) \mid X \in V\}$ and $[V]_{\leq m} = \{val(X) \mid X \in V, |X| \leq m\}$.

A grammar is called α-balanced iff $|X_\ell|$ and $|X_r|$ are α-balanced for any variable $X \in V$ with $X \to X_\ell X_r$. For any variable X of an α-balanced grammar, the height of X is $O(\log |X|)$. In particular, the height of an α-balanced grammar is $O(\log N)$, where N is the length of the string represented by the grammar.

The next lemma is well known for random access and substring expansion on grammar compressed strings.

Lemma 1 (Substring expansion). *Suppose that for every variable in V we store the length of the string it derives. Then, we can expand any substring of length l of $val(X)$ for any variable $X \in V$ in $O(h+l)$ time, where h is the height of X.*

As a corollary of this lemma, for balanced grammars we can support substring expansion in $O(\log |X| + l)$ time.

[2] Of course, we ignore any trivial input string of length one or zero.

2.3 LZ78-style Grammar Compression

We describe the basic LZ78 [17] algorithm and LZD (a variant of LZ78) [4]. LZ78 algorithm transforms an input string S of length N into a sequence $P_1, P_2, \ldots, P_n$ of substrings such that each *phrase* $P_k = p_{k_1}p_{k_2}$ is defined as follows. If $k = 1$, $p_{k_1} = \varepsilon$ and $p_{k_2} = S[1]$. For $k \geq 2$, p_{k_1} is the longest prefix of $S[|P_1 \cdots P_{k-1}| + 1..N]$ in $\{P_1, \ldots, P_{k-1}\}$ if it exists or $p_{k_1} = \varepsilon$ otherwise, and p_{k_2} is the next symbol $S[|P_1 \cdots P_{k-1}p_{k_1}| + 1]$. For example, the *Fibonacci string* $S = abaababaabaab$ is parsed into the sequence $(P_1, P_2, P_3, P_4, P_5, P_6, P_7) = (a, b, aa, ba, baa, baab, c)$. A recursive expansion of LZ78 phrase is directly interpreted into a derivation in grammar compression associating a variable X_k with a phrase $P_k = p_{k_1}p_{k_2}$ by the production rule $X_k \to X_{k_1}X_{k_2}$. Finally, the righthand of the start production rule contains all the variables associated with pairs. For example, a grammar associated with the LZ78 phrases for $S = abaababaabaabc$ is defined as follows.

$$\left\{ \begin{array}{l} X_8 \to X_1X_2X_3X_4X_5X_6X_7,\ X_7 \to c,\ X_6 \to X_5b,\ X_5 \to X_4a, \\ X_4 \to X_2a,\ X_3 \to X_1a,\ X_2 \to b,\ X_1 \to a \end{array} \right\}$$

Given this correspondence between LZ78 phrases and production rules, hereafter we can regard the output of LZ78 algorithm as a grammar compression rather than as a sequence of LZ78 phrases.

LZD (*LZ-Double*) parsing for S is a variant of LZ78 defined by a sequence of phrases $P_k = p_{k_1}p_{k_2}$ as follows. p_{k_1} and p_{k_2} are in $\{P_1, \ldots, P_{k-1}\} \cup \Sigma$, and p_{k_1} is the longest prefix of $S[|P_1 \cdots P_{k-1}| + 1..N]$, and p_{k_2} is the longest prefix of $S[|P_1 \cdots P_{k-1}p_{k_1}| + 1..N]$. Intuitively, P_k in LZD is a concatenation of two longest prefixes of unprocessed S appearing in the generated phrases for the processed S. For the same $S = abaababaabaabc$, the LZD phrases $(P_1, P_2, P_3, P_4) = (ab, aab, abaab, aabc)$ defines the following grammar.

$$\{X_5 \to X_1X_2X_3X_4,\ X_4 \to X_2c,\ X_3 \to X_1X_2,\ X_2 \to aX_1,\ X_1 \to ab\}$$

Practically, the number of LZD phrases, $|LZD|$, is smaller than $|LZ78|$, however, there exists a worst case satisfying $|LZD| = \Omega(N^{\frac{1}{3}})$ and requires $\Omega(N^{\frac{5}{4}})$ computation time as reported in [1].

3 LZ-ABT: Online α-balanced Grammar Compression

In this section, we present an online α-balanced grammar compression algorithm, named LZ-ABT. Let S (resp. V) always denote the current state of the input string (resp. the set of variables). While appending a character to a string S, we incrementally add α-balanced variables to V in a greedy manner. So $N := |S|$ and $n := |V|$ are monotonically non-decreasing.

We remark that we will not "complete" an α-balanced grammar so as to always prepare for new characters to come. Instead, we maintain a tail string S_{tail} and a stack that stores a sequence $S_1, S_2, \ldots, S_t$ of strings such that

1. $S_j \in [V]$ for any $j < t$,
2. $S_t \in [V]$ or a single character,
3. S_{tail} is a prefix of some string in $[V]$ or a single character,
4. $S_1 S_2 \cdots S_t S_{tail} = S$,
5. $|S_1| > |S_2| > \cdots > |S_t|$, and
6. any pair of adjacent strings in the stack is not α-balanced.

We note that we do not actually store the strings $S_1, S_2, \ldots, S_t, S_{tail}$, but we just remember the corresponding variable that derives each string (or a single character), and for S_{tail} additionally the length $|S_{tail}|$. Thus, $S_1, S_2, \ldots, S_t, S_{tail}$ together with V represent S in compressed space of $O(n+t) = O(n)$ due to the following lemma.

Lemma 2. *At any moment of our LZ-ABT algorithm, the stack size t is bounded by $O(\log N)$.*

Proof. Since any pair of adjacent strings in the stack is not α-balanced and monotonically shrinking, $N > |S_1| > \beta|S_2| > \beta^2|S_3| > \cdots > \beta^{t-1}|S_t| > \beta^{t-1}$. As $\beta = (1-\alpha)/\alpha > 1$ is constant, $t \leq \log_\beta N + 1 = O(\log N)$. □

LZ-ABT consists of two sub-procedures `parse` and `merge`, which are executed alternatingly. In `parse`, we work on S_{tail} and search for the longest prefix p of S_{tail} that matches a string in $[V]_{\leq |S_t|\beta}$. Here the length limit $|S_t|\beta$ is set to avoid the situation where p is too large to merge with S_t under the condition of α-balanced grammar. As soon as we find that p cannot be longer by appending characters to S_{tail}, we push p into the stack and execute `merge`. Note that at the moment p is pushed, the condition for the top of the stack could be broken, i.e., S_{t-1} could be shorter than S_t and/or α-balanced with S_t. This is mended in `merge`, in which we iterate the following procedure while the two strings S_{t-1} and S_t on the top of the stack are α-balanced: pop two strings (let S_ℓ and S_r be the popped strings in the order of positions) to make a new variable $X \to X_\ell X_r$ with $val(X_\ell) = S_\ell$ and $val(X_r) = S_r$, and then push $val(X)$.

Algorithm 1 shows our algorithm, in which `parse` is described in Line 4–6 and `merge` is described in Line 8–12. In Subsect. 3.1, we will study the behavior of LZ-ABT that is varied according to the parameter α. In Subsect. 3.2, we will present a standard implementation of `parse` based on Patricia trees and its analysis. Finally in Subsect. 3.3, we show how to execute the algorithm efficiently in compressed space.

3.1 Behavior of LZ-ABT with Varying α

First of all, we remark that LZ-ABT does not work as we expect if α is too large. The next lemma shows valid α by which it is guaranteed that S_t cannot be too large to merge with S_{t-1} under the condition of α-balanced grammar.

Lemma 3. *If $0 < \alpha \leq (3-\sqrt{5})/2$, at any moment of LZ-ABT algorithm, the following holds: S_{t-1} and S_t are α-balanced iff $|S_{t-1}| \leq |S_t|\beta$.*

Algorithm 1. Construction of LZ-ABT's grammar of an input string

Data: S: current input string. V: current set of variables of grammar. $\mathtt{S}$: current stack of size t. S_{tail}: current tail string. α: an input parameter of positive real number less than $(3-\sqrt{5})/2$. Let $\beta = (1-\alpha)/\alpha$. For convenience, let $|S_0| = 0$.

Output: Report V, $\mathtt{S}$ and S_{tail} as a compressed representation of S.

```
1  initialize V ← ∅, S ← ∅, S_tail ← ε;
2  while receive a new character a to append do
3      S_tail ← S_tail a, S ← Sa;
4      while S_tail is not prefixed by any string in [V]_{≤|S_t|β} do
5          p ← longest prefix of string S_tail in [V]_{≤|S_t|β} ∪ Σ;
6          S_tail ← S_tail[|p|+1..|S_tail|] ;          // truncate p from S_tail
7          S.push(p);
8          while t > 1 and (|S_{t-1}|β ≤ |S_t|) do
9              S_r ← S.pop();
10             S_ℓ ← S.pop();
11             add a new variable X deriving S_ℓ S_r into V;
12             S.push(S_ℓ S_r);
13     report V, S and S_tail;
```

Proof. It suffices to show that a pushed string S_t cannot be too long (i.e., $|S_t| \leq |S_{t-1}|\beta$ for a constant $\beta = (1-\alpha)/\alpha$). There are two situations where a string is pushed to the stack; one is in `parse` and another in `merge`. For the former situation, we are sure that $|S_t| \leq |S_{t-1}|\beta$ by the limit we set for the prefix search. For the latter situation, recall that the pushed string S_t $(= p)$ is derived by $X \to X_\ell X_r$. Let $val(X_\ell) = S_\ell$ and $val(X_r) = S_r$. Since S_{t-1} and S_ℓ were adjacent in the stack before S_ℓ and S_r were popped, $|S_{t-1}| > |S_\ell|\beta$ holds. Also, $|S_r| \leq |S_\ell|\beta < |S_{t-1}|$ holds because S_ℓ and S_r are α-balanced. Due to $|S_t|/|S_{t-1}| = (|S_\ell| + |S_r|)/|S_{t-1}| < |S_\ell|/|S_{t-1}| + 1 < 1/\beta + 1$, it suffices to show $1/\beta + 1 \leq \beta$ to see if $|S_t| \geq |S_{t-1}|\beta$ holds.

$$
\begin{aligned}
1/\beta + 1 &\leq \beta \\
\alpha/(1-\alpha) + 1 &\leq (1-\alpha)/\alpha \\
\alpha^2 - 3\alpha + 1 &\geq 0.
\end{aligned}
$$

Solving this formula under $0 < \alpha < 1/2$, we get $0 < \alpha \leq (3-\sqrt{5})/2$, and hence, the statement holds. □

Next we show that if the balancing condition is stronger than a certain point, LZ-ABT generates a grammar whose derivation tree is a perfect binary tree in an online manner (partially constructed subtrees are stacked in the right place), and thus, essentially works as an online algorithm of Bisection [11].

Lemma 4. *If $\alpha > 1/3$ (or equivalently $\beta = (1-\alpha)/\alpha < 2$), LZ-ABT generates a grammar whose derivation tree is a perfect binary tree in an online manner.*

Proof. We say that a string S_i for any $1 < i \leq t$ in the stack is *well-placed* iff $|S_i| = 2^k$ for some integer $k \geq 0$ and $\sum_{j=1}^{j<i} |S_j|$ is divisible by 2^k. We show that the length of any string in $[V]$ is a power of two and $S_1, S_2, \ldots, S_t$ are well-placed by induction. We assume that the condition holds right before pushing a string into the stack in `parse`, and show that the condition still holds after finishing the succeeding `merge`.

By the inductive assumption, the length of the pushed string S_t $(= p)$ is a power of two. In addition, by the limit we set for the prefix search, it holds that $|S_t| \leq |S_{t-1}|\beta < 2|S_{t-1}|$, and thus, $|S_t|$ is a power of two that is at most $|S_{t-1}|$. If $|S_t| < |S_{t-1}|$, `merge` does not merge S_{t-1} and S_t because S_{t-1} and S_t are not α-balanced for $(1-\alpha)/\alpha < 2$ due to $|S_{t-1}|/|S_t| \geq 2 > \beta$. By the condition $S_1 > S_2 > \cdots > S_t$, $|S_i|$ is divisible by $|S_t|$ for any $i < t$, and thus, S_t is well-placed. If $|S_t| = |S_{t-1}| = 2^k$, two strings are popped and the merged string of length 2^{k+1} is pushed (being new S_t). At this moment, the situation is exactly the same as before, i.e., the inductive assumption is still valid for $S_1, S_2, \ldots, S_{t-1}$ and $|S_t|$ is a power of two that is at most $|S_{t-1}|$. Therefore, it is proved that S_t is well-placed in the same way. Since `merge` just repeats this procedure, we see that $S_1, S_2, \ldots, S_t$ are always well-placed and every merged string has a length of power of two. □

3.2 Implementation with Patricia Trees and Its Analysis

The Patricia tree $\mathcal{T}_V$ of $[V]$ is the compacted trie storing the strings in $[V]$. Each edge label (string) is stored by a pointer to input string in constant space. In order to compute the longest prefix p of S_{tail} that matches a string in $[V]_{\leq |S_t|\beta}$, we only have to traverse the tree from the root by $S_{tail}[1..|S_t|\beta]$ as long as possible. Suppose that we stop at a (possibly implicit) node v during the traversal, the lowest node that represents a string in $[V]_{\leq |S_t|\beta}$ on the path from the root to v is actually representing p. We can maintain v and p online while appending new character to S. Let pp' be the string v represents. The cost of the traversal is $O(|pp'| \log \sigma)$. When p is parsed out, $O(|p'| \log \sigma)$ is considered to be the "overrun" cost. as we restart the next parsing with p'. We can charge this cost to $O(|p|\beta \log \sigma) = O(|p| \log \sigma)$ because, by definition of α-balanced grammars, there is a string $pq \in [V]_{\leq |S_t|\beta}$ such that p' is a prefix of q, and p and q are α-balanced. Therefore, the cost of `parse` is bounded by $O(N \log \sigma)$ in total.

Next, we consider how to update $\mathcal{T}_V$ when a new phrase $S_\ell S_r$ is added in `merge`. We simply traverse $\mathcal{T}_V$ from the root by $S_\ell S_r$ as long as possible, and branch out a new edge.[3] The total cost of maintaining $\mathcal{T}_V$ is bounded by $O(L \log \sigma)$, where L is the sum of the lengths of strings in $[V]$. Since the grammar is α-balanced, for every position i in S, $S[i]$ is derived by at most $O(\log N)$ variables, and hence, $L = O(N \log N)$.

[3] Since S_ℓ is represented in $\mathcal{T}_V$, we can shortcut by starting the traversal from the node representing S_ℓ, but it does not change the complexity.

In summary, we obtain the following theorem.

Theorem 1. *Given a string of length N over an alphabet of size σ online, LZ-ABT can be computed in $O(N \log N \log \sigma)$ time and $O(n)$ working space in addition to $N \log \sigma$ bits of space for the input, where n is the size of the grammar to output.*

3.3 LZ-ABT in Compressed Space

We show how to execute our LZ-ABT algorithm described in Subsect. 3.2 in compressed space: in $O(N \log^2 N \log \sigma)$ time and $O(n)$ space, or $O(N \log N \log \sigma)$ time and $O(n \log^* N)$ space.

The idea is to retrieve the edge labels of the Patricia tree $\mathcal{T}_V$ from the α-balanced grammar rather than referring to S explicitly stored in memory. Since each edge label is a substring of $val(X)$ for some variable X, we can retrieve the edge label by Lemma 1 if we remember a triple (X, b, l) such that $val(X)[b..b + l - 1]$ is the edge label. This gives us the following theorem:

Theorem 2. *Given a string of length N over an alphabet of size σ online, LZ-ABT can be computed in $O(N \log^2 N \log \sigma)$ time and $O(n)$ working space, where n is the size of the grammar to output.*

In order to accelerate the algorithm, we can utilize the bookmarking data structure of [3] on balanced grammars. We describe the basic idea of the data structure as follows:

Lemma 5. *([3]). Let X be a variable whose derivation tree is balanced. If we construct a bookmarking data structure for a position b in X using $O(\log^* |X|)$ space, we can later expand, given any l, $val(X)[b..b + l - 1]$ in $O(l)$ time.*

Proof. For a balanced grammar, we can choose (at most) two nodes of height $O(\log g)$ that covers $val(X)[b..b+g]$ for any g. In the preprocessing phase we compute such nodes for every $g \in \tau(|X|)$, and remember their node labels (variables) in $O(\log^* |X|)$ space.

In the query phase, given l, we first choose the smallest $g \in \tau(|X|)$ with $g \geq l$ in $O(\log^* l) = O(l)$ time. If there is no such g, it holds that $\log |X| < l$, and thus, $val(X)[b..b + l - 1]$ is expanded in $O(\log |X| + l) = O(l)$ time by Lemma 1. Now suppose that g exists, and Y and Z are the variables we remember for g. Notice that the heights of Y and Z are both $O(l)$ by definition. Since $val(X)[b..b + l - 1]$ is a substring of $val(Y)val(Z)$, we can expand it (suppose that we also remember the relative position of b in $val(Y)$) by using Lemma 1 in $O(\log |Y| + \log |Z| + l) = O(l)$ time. □

By Lemma 5, for every edge label associated with a triple (X, b, l), we can bookmark b in X so that later we can expand the edge label in time linear to its length. In addition, we can also support *prefix expansion*, in which we want to expand $val(X)[b..b + l']$ in $O(l')$ time even if we do not know l' in advance. This can be done with a standard doubling technique: we gradually increase

the length of expansion starting from a small constant and double the length of expansion when it is needed.

The last issue we have to address is how to maintain bookmarking data structures in our online setting: In contrast to [3], in which the bookmarking positions are static, we have to add bookmarks when new edge labels appear. A naive approach would take $O(\log |X| \log^* |X|)$ time to compute, for every $g \in \tau(|X|)$, two nodes of height $O(\log g)$ that covers $val(X)[b..b+g]$.

To solve this issue, we slightly modify the data structure. First of all, we bookmark the first position for every variable X to support prefix expansion of the string derived by the variable. When a new variable X is created, we do the following spending $O(\log |X|)$ time: we traverse down along the leftmost path of the derivation tree of X and compute, for each $g \in \tau(|X|)$, the lowest node that covers $val(X)[1..g]$. Storing the labels of these nodes is enough to support prefix expansion of $val(X)$. In a completely symmetric way, we also bookmark the last position for every variable X to support "suffix" expansion.

When we bookmark a new position b in X, we do the following spending $O(\log |X|)$ time: we traverse the derivation tree of X from the root to the position b, and identify, for each $g \in \tau(|X|)$, the lowest node v_g that covers $val(X)[b..b+g]$, and remember its label. Suppose that, in the query phase, we want to expand $val(X)[b..b+l-1]$ for l such that there exists $g \in \tau(|X|)$ with $g \geq l$. Since $val(X)[b..b+l-1]$ is a concatenation of a suffix of the left child of v_g and a prefix of right child of v_g, we can expand it in $O(l)$ time using the bookmarking data structures for prefix/suffix expansion created for every variable.

In summary, we have shown that we can add a bookmark in $O(\log N)$ time in our online setting. Since we need $O(n)$ bookmarks, we use $O(n \log^* N)$ space, and the total cost $O(n \log N)$ of constructing bookmarking data structures is dominated by the cost $O(N \log N \log \sigma)$ for prefix expansion of edge labels. Therefore, we get the following theorem:

Theorem 3. *Given a string of length N over an alphabet of size σ online, LZ-ABT can be computed in $O(N \log N \log \sigma)$ time and $O(n \log^* N)$ working space, where n is the size of the grammar to output.*

4 Experiments

We implemented in C++ LZ-ABT described in Theorem 1 of Sect. 3.2, which runs in $O(N \log N \log \sigma)$ time and $O(N)$ space. To confirm its empirical performance, we examined the compression ratio, compression speed, and working space with several $\alpha \in \{0.1, 0.2, 0.3\}$ compared with a practical implementation of LZD[4] in $O(n(M + \min(m, M) \log \sigma))$ time and $O(N)$ space where M is the maximal length of factors. From well-known datasets for benchmark, we chose 6 general texts (`dblp.xml.200MB`, dna.200MB, `english.200MB`, `pitches.50MB`, `proteins.200MB`, `sources.200MB`)[5] and 7 repetitive texts (`Escherichia_Coli`,

[4] https://github.com/kg86/lzd.
[5] http://pizzachili.dcc.uchile.cl/texts.html.

Table 1. LZ-ABT v.s. LZD: Compression ratio, throughput, and working space, where "compression ratio" denotes n/N for the number n of phrases and $N = |S|$, "throughput" denotes the number of characters ($\times 10^6$) per second, and "working space" denotes the required space in megabytes. The results written in bold represent the best results among the ones of LZD and LZ-ABT for $\alpha = 0.1, 0.2, 0.3$, that is, the smallest, largest, smallest numbers for the compression ratio, throughput, and working space, respectively.

Data set	$\|S\|$ [MB]	Compression ratio [%]				Throughput [Mchar/sec]				Working space [MB]			
		LZ-ABT			LZD	LZ-ABT			LZD	LZ-ABT			LZD
		$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 0.3$		$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 0.3$		$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 0.3$	
dblp.xml.200 MB	200.00	17.67	18.43	20.44	**15.72**	6.26	5.73	5.15	**6.81**	1395.46	1444.63	1557.68	**842.29**
dna.200 MB	200.00	36.74	36.19	35.85	**34.25**	2.32	2.31	2.23	**2.59**	2650.67	2695.31	2747.39	**1874.78**
english.200 MB	200.00	33.01	**32.73**	34.54	33.91	**2.91**	2.40	2.09	2.45	2432.53	2462.47	2580.70	**1520.37**
pitches.50 MB	50.00	60.86	58.89	58.64	**58.61**	2.30	2.28	2.23	**2.57**	1113.54	1105.90	1117.55	**631.67**
proteins.200 MB	200.00	55.20	54.11	**53.65**	55.90	**1.92**	1.81	1.62	1.91	4208.49	4260.38	4320.35	**2417.78**
sources.200 MB	200.00	32.07	31.99	34.82	**29.54**	3.14	3.59	3.12	**4.01**	2381.82	2414.60	2574.88	**1384.26**
Escherichia_Coli	107.47	12.68	12.02	**11.76**	14.80	**8.71**	8.68	8.56	7.83	638.29	634.02	643.61	**447.56**
cere	439.92	3.23	**2.95**	2.98	4.21	**27.00**	26.44	24.95	22.33	982.80	968.54	985.22	**857.34**
einstein.en.txt	445.96	**0.13**	**0.13**	0.14	0.16	**231.59**	230.31	220.68	229.13	472.06	472.08	478.55	**463.64**
influenza	147.64	4.33	**4.21**	4.34	4.32	24.68	23.64	21.26	**26.83**	414.26	416.55	429.03	**300.55**
kernel	246.01	1.76	**1.72**	1.87	2.81	**63.91**	62.96	55.98	44.59	404.51	404.98	417.52	**389.68**
para	409.38	4.54	4.19	**4.17**	5.91	20.04	**20.28**	19.62	16.53	1054.00	1037.84	1054.77	**963.78**
world_leaders	44.79	2.88	**2.78**	3.05	3.27	50.69	48.51	41.25	**54.02**	107.47	97.02	109.70	**79.40**
lz78s_large_ratio.txt	516.01	**0.00**	**0.00**	**0.00**	0.24	**334.25**	316.23	304.75	101.21	**517.24**	517.28	517.32	568.01
lz78s_worst_time2.txt	0.01	13.35	12.77	15.85	**8.90**	1.06	1.15	1.11	**1.34**	0.20	0.20	0.20	**0.17**
lz78s_worst_time3.txt	120.63	0.10	0.10	0.11	**0.04**	**251.07**	241.77	226.24	113.07	127.13	127.37	128.05	**122.26**

`cere`, `einstein.en.txt`, `influenza`, `kernel`, `para`, `world_leaders`)[6]. In addition, we show a comparison of these algorithms for the following anomalous strings reported in [1] as a worst case for LZD parsing.

- `lz78s_large_ratio.txt`: A string requiring $\Omega(N^{\frac{1}{3}})$ phrases for LZD;
- `lz78s_worst_time2.txt`: A string from k^2 different alphabet symbols (e.g., $k = 4$) requiring $\Omega(N^{\frac{5}{4}})$ time for LZD;
- `lz78s_worst_time3.txt`: A string from a binary alphabet requiring $\Omega(N^{\frac{5}{4}})$ time for LZD.

These worst case strings are created using the program available from a public resource[7]. We compiled the programs with g++-7.2.1 using -Ofast option and examined the performance on one core of an 8-core Intel(R) Xeon(R) CPU E7-8837 (2.66 GHz) Linux (CentOS6) machine with 1TB memory.

Table 1 shows the compression ratio (n/N), throughput (10^6 chars per sec.) and working space (MB) of LZ-ABT and LZD for the indicated benchmarks. For each $\alpha \in \{0.1, 0.2, 0.3\}$, the compression ratio and throughput of LZ-ABT are very competitive to LZD for general texts, and they are better for repetitive texts. Especially, LZ-ABT avoids the worst case behavior of LZ78-style grammar compression for the anomalous strings. On the other hand, LZ-ABT requires larger working space than LZD for all texts. This is because the implementation of LZD stores only all phrases in a Patricia tree but our implementation stores decompressed strings of all variables in addition to all phrases in a Patricia tree. We believe that this drawback will be solved by implementing a Patricia tree in compressed form as described in Sect. 3.3.

5 Conclusions and Future Work

We proposed a novel LZ78-style compression algorithm, LZ-ABT, that is a first practical implementation of grammar compression preserving the α-balancedness. We implemented LZ-ABT in uncompressed space and showed its empirical performance. As a result, the compression ratio and time were very competitive to LZD, and especially for repetitive texts, LZ-ABT showed better performance than LZD. Furthermore, we really confirmed the phenomenon reported in [1] and our algorithm avoid the worst case running time. On the other hand, unfortunately, we observed the increase of working space for almost all benchmarks. In addition, we introduced several ideas to implement LZ-ABT in compressed space and showed related results in Sect. 3.3. An improvement of LZ-ABT in compressed space is one of important future works in this study.

[6] http://pizzachili.dcc.uchile.cl/repcorpus/real/.
[7] https://bitbucket.org/dkosolobov/lzd-lzmw.

In terms of theoretical bounds for approximation ratios to the smallest grammar, it is open whether LZ-ABT makes difference to those of LZD, which was proved to be $\Omega(N^{1/3})$ and $O((N/\log N)^{2/3})$ in [1]. It seems not difficult to get $O((N/\log N)^{2/3})$ in a similar way to [1,2]. Meanwhile, if LZ-ABT works as an online algorithm of Bisection, we have the tight bound $\Theta((N/\log N)^{1/2})$ of [5].

Acknowledgments. This work was supported by JST CREST (Grant Number JPMJCR1402), and KAKENHI (Grant Numbers 18K18111, 17H01791 and 16K16009).

References

1. Badkobeh, G., Gagie, T., Inenaga, S., Kociumaka, T., Kosolobov, D., Puglisi, S.J.: On two LZ78-style grammars: compression bounds and compressed-space computation. In: Fici, G., Sciortino, M., Venturini, R. (eds.) SPIRE 2017. LNCS, vol. 10508, pp. 51–67. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67428-5_5
2. 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)
3. 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
4. Goto, K., Bannai, H., Inenaga, S., Takeda, M.: *LZD Factorization*: simple and practical online grammar compression with variable-to-fixed encoding. In: Cicalese, F., Porat, E., Vaccaro, U. (eds.) CPM 2015. LNCS, vol. 9133, pp. 219–230. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19929-0_19
5. 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
6. Jez, A.: Approximation of grammar-based compression via recompression. Theor. Comput. Sci. **592**, 115–134 (2015)
7. Jez, A.: A really simple approximation of smallest grammar. Theor. Comput. Sci. **616**, 141–150 (2016)
8. Larsson, N.J., Moffat, A.: Offline dictionary-based compression. In: Data Compression Conference, DCC 1999, pp. 296–305 (1999)
9. Lohrey, M.: Algorithmics on SLP-compressed strings: a survey. Groups Complex. Cryptol. **4**(2), 241–299 (2012)
10. Miller, V.S., Wegman, M.N.: Variations on a theme by Ziv and Lempel. In: Apostolico, A., Galil, Z. (eds.) Combinatorial Algorithms on Words. NATO ASI Series, vol. 12, pp. 131–140. Springer, Heidelberg (1985)
11. Nelson, G., Kieffer, J., Cosman, P.: An interesting hierarchical lossless data compression algorithm. In: IEEE Information Theory Society Workshop (1995)
12. Nevill-Manning, C.G., Witten, I.H.: Identifying hierarchical strcture in sequences: a linear-time algorithm. J. Artif. Intell. Res. (JAIR) **7**, 67–82 (1997)
13. Rytter, W.: Application of Lempel-Ziv factorization to the approximation of grammar-based compression. Theor. Comput. Sci. **302**(1–3), 211–222 (2003)
14. Sakamoto, H.: A fully linear-time approximation algorithm for grammar-based compression. J. Discrete Algorithms **3**(2–4), 416–430 (2005)

15. Storer, J.A., Szymanski, T.G.: The macro model for data compression (extended abstract). In: Proceedings of the 10th Annual ACM Symposium on Theory of Computing, pp. 30–39 (1978)
16. Takabatake, Y., I, T., Sakamoto, H.: A space-optimal grammar compression. In: Proceedings of ESA 2017, pp. 67:1–67:15 (2017)
17. Ziv, J., Lempel, A.: Compression of individual sequences via variable-length coding. IEEE Trans. Inf. Theory **24**(5), 530–536 (1978)

Faster Coreset Construction for Projective Clustering *via* Low-Rank Approximation

Rameshwar Pratap[1(✉)] and Sandeep Sen[2]

[1] Wipro Technologies, Bangalore, India
rameshwar.pratap@gmail.com
[2] IIT Delhi, New Delhi, India
ssen@cse.iitd.ernet.in

Abstract. In this work, we present a randomized coreset construction for projective clustering, which involves computing a set of k closest j-dimensional linear (affine) subspaces of a given set of n vectors in d dimensions. Let $A \in \mathbb{R}^{n \times d}$ be an input matrix. An earlier deterministic coreset construction of Feldman *et. al.* [10] relied on computing the SVD of A. The best known algorithms for SVD require $\min\{nd^2, n^2d\}$ time, which may not be feasible for large values of n and d. We present a coreset construction by projecting the matrix A on some orthonormal vectors that closely approximate the right singular vectors of A. As a consequence, when the values of k and j are small, we are able to achieve a faster algorithm, as compared to [10], while maintaining almost the same approximation. We also benefit in terms of space as well as exploit the sparsity of the input dataset. Another advantage of our approach is that it can be constructed in a streaming setting quite efficiently.

1 Introduction

Succinct representation of Big data – Coreset: Recent years have witnessed a dramatic increase in our ability to collect data from various sources. This data flood has surpassed our ability to understand, analyse and process them. *Big data* is a new terminology that has become quite popular in identifying such datasets that are difficult to analyse with the current available technologies. One possible approach to manage such large volume of datasets is to keep a succinct summary of the datasets such that it approximately preserves the required properties of the original datasets. This notion was initially formalised by Agarwal *et al.* [1], and they coined the term *coreset* for such summaries. Intuitively, a coreset can be considered as a semantic compression of the input. For example: in the case of clustering, a coreset is a weighted subset of the data such that the cost of a clustering algorithm evaluated on the coreset closely approximates to the corresponding cost on the entire dataset. Consider a set Q (possibly of infinite size)

R. Pratap—This work done when author was affiliated with TCS Innovation Labs.

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 336–348, 2018.
https://doi.org/10.1007/978-3-319-94667-2_28

of query shapes (for example: subspaces, set of points, set of lines etc.), then for every shape $q \in Q$, the sum of distances from q to the input points, and the sum of distances from q to the points in the coreset, is approximately the same. If the query set belongs to some particular candidate query set, then such coreset is called as a *weak* coreset [15]; and if the coreset approximates the distances from all possible (potentially infinite) query shapes, then it is called as *strong* coreset. Coresets are a practical and flexible tool which require no or minimal assumption on the data. Although the analysis techniques for coreset construction are a bit involved, and require tools from computational geometry and linear algebra, the resulting coreset construction algorithms are easy to implement. Another important property of coresets is that they can be constructed in a streaming and distributed setting quite efficiently. This is due to the fact that unions of coresets are coresets, and coresets of coresets are also coresets [12]. Also, using these properties it is possible to construct coresets in a tree-wise fashion which can be parallelized in a Map-Reduce style [10].

Coreset constructions have been studied extensively for various data analysis tasks. There are usually two steps involved in the coreset construction – dimensionality reduction, and cardinality reduction. The dimension reduction step of the coreset construction includes projecting points in a low dimension space such that the original geometry of points is also preserved in the low dimension. These projection techniques includes SVD decomposition, random projections, row/column subset selections, or any combinations of these (see [5,10]). The cardinality reduction step includes contracting the input size *via* sampling or other geometric analysis approach on the reduced dimension instance of the input. We refer readers to survey articles of Jeff M. Phillips [16] and Agarwal *et al.* [2].

In this work, we focus on the dimension reduction step of coreset construction for the projective clustering problem. In the paragraph below, we discuss the motivation behind the projective clustering problem.

Projective clustering: Clustering is one of the most popular techniques for analyzing large data, and is widely used in many areas such as classification, unsupervised learning, data mining, indexing, pattern recognition. Many popular clustering algorithms such as k-means, BIRCH [19], DBSCAN [6] are full dimensional – they give equal importance to all the dimensions while computing the distance between two points. These clustering algorithms works well in low dimensional datasets, however, due to the *"curse of dimensionality"* such algorithms scale poorly in high dimensions. Moreover, in high dimensional datasets a full dimensional distance might not be appropriate as farthest neighbour of a point is expected to be roughly as close as its nearest neighbour [14]. These problems are often handled *via* methods such as Principal component analysis (PCA) or Johnson-Lindenstrauss lemma by finding a low dimensional representation of the data obtained by projecting all points on a subspace so that the information loss is minimized. However, projecting all the points in a single low dimensional subspace may not be appropriate when different clusters lie in different subspaces. This motivates the study of projective clustering which involves finding clusters along different subspaces. Projective clustering algorithms have been widely applicable for indexing and pattern discovery in high dimensional datasets.

1.1 Our Contribution

With the above motivation we study the dimension reduction step of coreset construction for projective clustering problem. We first briefly describe the subspace and projective clustering problems. In a j-subspace clustering problem, given a set of n, d dimensional vectors, denoted by $A \in \mathbb{R}^{n\times d}$, the problem is to find a j-dimensional subspace such that it minimizes the sum of squared distances from the rows of A, over every j-dimensional subspace. Further, in the problem of linear (affine) (k, j)-projective clustering, the goal is to find a closed set $\mathcal{C}$ which is the union of k linear (affine) subspaces each of dimension j, such that it minimizes the sum of squared distances from the rows of A, over every possible choice of $\mathcal{C}$ (see Definitions 8 and 9).

Feldman *et al.* [10] presented a deterministic coreset construction for these clustering problems. Their coreset construction relies on projecting the rows of A on the first few right singular values of A. However, the main drawback of their construction is that it requires computing the SVD of A which is expensive for large values of n and d. Cohen *et al.* [5] suggested *"projection-cost-preserving-sketch"* for various clustering problems. Their sketches are essentially the dimensionality reduction step of the coreset construction. Using a low rank approximation of A, they suggested a faster coreset construction for the subspace clustering problem. However, it was not clear that how their techniques could be extended for projective clustering problem. In this work, we extend their techniques and obtain a faster dimension reduction for projective clustering, and as a consequence, a faster coreset construction for the projective clustering problem. In Sect. 3, we first revisit the techniques for subspace clustering problem, and in Sect. 4 we present our coreset construction for projective clustering problem. We state our main result as follows: (In the following theorem, $\mathbf{nnz}(A)$ denotes the number of non-zero entries of A.)

Theorem 1. *Let $A \in \mathbb{R}^{n\times d}$, $\epsilon \in (0,1)$, and j, k be two integers less than $(d-1)$, and $(n-1)$ respectively such that $k(j+1) \leq d-1$. Then there is a randomized algorithm which outputs a matrix A^* of rank $O\left(k(j+1)/\epsilon^2\right)$ such that for every non-empty closed set $\mathcal{C}$, which is the union of k linear (affine) subspaces each of dimension at most j, the following holds w.h.p.*

$$\left|\left(\text{dist}^2(A^*,\mathcal{C}) + \Delta^*\right) - \text{dist}^2(A,\mathcal{C})\right| \leq \epsilon \text{dist}^2(A,\mathcal{C}).$$

where, $j^* = k(j+1)$; $\Delta^* = ||A - A^{O(\frac{j^*}{\epsilon^2})}||_F^2$; $\text{dist}^2(A,\mathcal{C})$ denotes the sum of squared distances from each row of A to its closest point in $\mathcal{C}$; and $A^{O(\frac{j^*}{\epsilon^2})}$ is the best rank $O(\frac{j^*}{\epsilon^2})$ approximation of A. The expected running time of the algorithm is $\tilde{O}\left(\mathbf{nnz}(A)\frac{j^*}{\epsilon^3} + (n+d)\frac{j^{*2}}{\epsilon^6} + \frac{ndj^*}{\epsilon^2}\right)$.[1]

Remark 1. We develop our coreset by projecting points on some orthonormal vectors that closely approximate the right singular vectors of A, and we obtain

[1] Here, $\tilde{O}$ is the asymptotic notation that ignores logarithmic factors.

them using the algorithm of Sarlós [17]. The expected running time of our algorithm is better than the corresponding deterministic algorithm of [10] when $n \geq d$ and $j^* = o(n)$, or, when $n < d$ and $j^* = o(d)$, where $j^* = k(j+1)$. Further, as the coreset construction time depends on the number of non-zero entries of the matrix, our algorithm is substantially faster for sparse data matrices. Please note that one can also use any other low-rank approximation algorithms such as [4] (instead of [17]), which offer multiplicative approximation guarantee. However, for completeness sake we use the bounds of [17], and compare our results with [10].

Remark 2. The term Δ^* is a positive constant, and is sum of squared singular values from $O(j^*/\epsilon^2)$ to d. We use A^* to approximately solve the clustering problem, and add the constant Δ^* in the clustering cost obtained from A^*, this sum gives a good approximation *w.r.t.* the cost of clustering on A.

Remark 3. An advantage of our coresets is that it can be constructed in the pass efficient streaming model [13], where access to the input is limited to only a constant number of sequential passes. We construct our coreset by projecting the matrix A on orthonormal vectors, that closely approximate the right singular vectors of A, our algorithm requires only two passes over the data in order to compute those orthonormal vectors using [17].

1.2 Related Work

Coreset construction has been studied extensively for the problem of j-subspace clustering. However, we will discuss a few of them that are more relevant to our work. Feldman *et al.* [7] developed a strong coreset whose size is exponential in d, j, logarithmic in n, and their coreset construction requires $O(n)$ time. Feldman *et al.* [9] improved their earlier result [7] and developed a coreset of size logarithmic in n, linear in d, and exponential in j. However, the construction requires $O(ndj)$ time. In [8] Feldman and Langberg showed a coreset construction of size polynomial in j and d (independent of n). Feldman *et al.* [10] presented a novel coreset construction for subspace and projective clustering. They showed that the sum of squared Euclidean distance from n rows of $A \in \mathbb{R}^{n \times d}$ to any j-dimensional subspace can be approximated upto $(1+\epsilon)$ factor, with an additive constant which is the sum of a few last singular values of A, by projecting the points on the first $O(j/\epsilon)$ right singular vectors of A. Thus, they were able to show the dimension reduction from d to $O(j/\epsilon)$. They also showed $O(k(j+1)/\epsilon^2)$ dimension reduction for (k, j)-projective clustering problem. Recently, for j-subspace clustering, Cohen *et al.* [5] improved the construction of [10] using only first $\lceil j/\epsilon \rceil$ right singular vectors, which is an improvement over [10] by a constant factor.

Sariel Har-Peled [11] showed that for projective clustering problem it is not possible to get a strong coreset of size sublinear in n, even for a simpler instance such as a family of pair of planes in $\mathbb{R}^3$. However, in a restricted setting, where points are on an integer grid, and the largest coordinate of any point is bounded

by a polynomial in n and d, Varadarajan *et al.* [18] showed that a sublinear sized coreset construction for projective clustering.

Organization of the paper: In Sect. 2, we present the necessary notations, definitions and linear algebra background that are used in the various proofs in the paper. In Sect. 3, we revisit the result of [5], and discuss the coreset construction for subspace clustering using their techniques. In Sect. 4, we extend the result of Sect. 3, and present the coreset construction for projective clustering problem. We conclude our discussion, and state some open questions in Sect. 5.

2 Preliminaries

Below we present some necessary linear algebra background. We first present some basic properties of Frobenius norm of a matrix. We define SVD (singular value decomposition) of a matrix, and its basic properties. We describe the expression about the distance of a point, and sum of square distances of the rows of matrix - from a subspace and a closed set.

Notations	
$A = U\Sigma V^T$	Columns of U, V are orthonormal and called as left and right singular vectors of A; $[\Sigma]$ is a diagonal matrix having the corresponding singular values
$A^{(m)} = U\Sigma^{(m)}V^T$	$\Sigma^{(m)}$ is the diagonal having the m largest entries of Σ, and 0 otherwise
$[X]_{d\times j}$	j orthonormal columns represent a j-dimensional subspace L in $\mathbb{R}^d$
$[X^{\perp}]_{d\times(d-j)}$	$(d-j)$ dimensional subspace $L^{\perp}$ orthogonal to subspace L
$\pi_{\mathcal{S}}(A)$	matrix formed by projecting A on the row span of $\mathcal{S}$
$\pi_{\mathcal{S},k}(A)$	The best rank-k approximation of A after projecting its rows on the row span of $\mathcal{S}$
$A^{(k)}$	The best rank-k approximation of A
$\mathbf{nnz}(A)$	The number of non-zero entries of A

Fact 1 (Frobenius norm and its properties). *Let $A \in \mathbb{R}^{n\times d}$, then square of Frobenius norm of A is defined as the sum of the absolute squares of its elements, i.e. $||A||_F^2 = \Sigma_{i=1}^{n}\Sigma_{j=1}^{d}a_{i,j}^2$. Further, if $\{\sigma_i\}_{i=1}^{d}$ are singular values of A, then $||A||_F^2 = \Sigma_{i=1}^{d}\sigma_i^2$. Also, if $\mathrm{tr}(A)$ be the trace of the matrix A then $||A||_F^2 = \mathrm{tr}(A^TA)$.*

Fact 2. *Let AX be the projection of points of A on the j-dimensional subspace L represented by an orthonormal matrix X. We can also write the projection of the points in the rows of A to L as AXX^T, these projected points are still d-dimensional, but lie within the j-dimensional subspace. Further, $||AX||_F^2 = ||AXX^T||_F^2$.*

The Singular Value Decomposition: A matrix $A \in \mathbb{R}^{n\times d}$ of rank at most r can be written due to its SVD decomposition as $A = \Sigma_{i=1}^{r}\sigma_t u^{(i)}{v^{(i)}}^T$. Here, $u^{(i)}$ and $v^{(i)}$ are i-th orthonormal columns of U and V respectively, and $\sigma_1 \geq \sigma_2, \ldots \sigma_r \geq 0$. Also, ${u^{(i)}}^T A = \sigma_i {v^{(i)}}^T$, and $Av^{(i)} = \sigma_i u^{(i)}$ for $1 \leq i \leq r$. Further, the matrix $A^{(k)}$ that minimizes $||A - B||_F$ among all matrices B (of rank at most k) is given by $A^{(k)} = \Sigma_{i=1}^{k} Av^{(i)}{v^{(i)}}^T$ - *i.e.* by projecting A on the first k right singular vectors of A.

l_2 distances to a subspace: Let L be a j-dimensional subspace in $\mathbb{R}^d$ represented by an orthonormal matrix $X \in \mathbb{R}^{d\times j}$. Then, for a point $p \in \mathbb{R}^d$, $||p^T X||_F^2$ is the squares of the length of projections of the point p on the subspace L. Similarly, given a matrix $A \in \mathbb{R}^{n\times d}$, $||AX||_F^2$ is the sum of squares of the length of projections of the points (rows) of A on the subspace L. Let $L^\perp$ be the orthogonal complement of L represented by an orthonormal matrix $X^\perp \in \mathbb{R}^{d\times(d-j)}$. Then, $||AX^\perp||_F^2$ is the sum of squares of distances of the points of A from L.

l_2 distance to a closed set: Let $S \in \mathbb{R}^d$ be a closed set and p be a point in $\mathbb{R}^d$. We define the l_2 distance between p and S by $\text{dist}^2(p, S) := \min_{s\in S} \text{dist}^2(p, s)$, *i.e.*, the smallest distance between p and any element $s \in S$. If S consists of union of k, j-dimensional subspaces $L_1, \ldots, L_k$, then $\text{dist}^2(p, S)$ denotes the distance from p to the closest set S. Similarly, given a matrix $A \in \mathbb{R}^{n\times d}$, $\text{dist}^2(A, S) := \Sigma_{i=1}^{n}\text{dist}^2(A_{i^*}, S)$. Here, A_{i^*} denotes the ith row of A.

Pythagorean theorem: Let $A \in \mathbb{R}^{n\times d}$, L be a j-dimensional subspace in $\mathbb{R}^d$ represented by an orthonormal matrix $X \in \mathbb{R}^{d\times j}$, and $L^\perp$ be the orthogonal complement of the subspace L represented by an orthonormal matrix $X \in \mathbb{R}^{d\times d-j}$. Then by Pythagorean theorem we have $||A||_F^2 = ||AX||_F^2 + ||AX^\perp||_F^2$. Further, if $\mathcal{C}$ is a closed set spanned by X, then due to the Pythagorean theorem we have $\text{dist}^2(A, \mathcal{C}) = ||AX^\perp||_F^2 + \text{dist}^2(AXX^T, \mathcal{C})$. We will use the following fact in our analysis which hold true due to Pythagorean theorem.

Fact 3. *Let $A \in \mathbb{R}^{n\times d}$, and $X \in \mathbb{R}^{d\times j}$ be a matrix having first j right singular vectors of A as columns, then due to the Pythagorean theorem, we have* $||A - AXX^T||_F^2 = ||A||_F^2 - ||AXX^T||_F^2$.

In the following, we state some facts from elementary linear algebra which are required for deriving the correctness of our result.

Fact 4. *For a square matrix $M \in \mathbb{R}^{n\times n}$, $\text{tr}(M)$ is the sum of all its diagonal entries. Further, for matrices $A \in \mathbb{R}^{n\times d}, B \in \mathbb{R}^{d\times n}$ due to the cyclic property of the* tr *function, we have* $\text{tr}(AB) = \text{tr}(BA)$. *Also for square matrices $M, N \in \mathbb{R}^{n\times n}$, due to the linear property of the* tr *function:* $\text{tr}(M \pm N) = \text{tr}(M) \pm \text{tr}(N)$.

Fact 5. *A symmetric matrix $M \in \mathbb{R}^{n\times n}$ is positive semidefinite if $x^T Mx > 0$ for all $x \in \mathbb{R}^n$. A matrix M is positive semidefinite then the following two statements are equivalent:*

- *there is a real nonsingular matrix N such that $M = N^T N$,*
- *all eigenvalues of M are nonnegative.*

Fact 6. *Let $A \in \mathbb{R}^{n\times d}$ and $U\Sigma V^T$ be the SVD of A. Then, the first j columns of V span a subspace that minimizes the sum of squares distances of the vectors in A from all j-dimensional subspace, and this sum is $\Sigma_{i=j+1}^{d}\sigma_i^2$. Thus, for any j-dimensional subspace represented by an orthonormal matrix X, we have $||AX^{\perp}||_F^2 \geq \Sigma_{i=j+1}^{d}\sigma_i^2$.*

Fact 7. *Let $M \in \mathbb{R}^{d\times l}$ be a matrix. Then, for an orthonormal matrix $X \in \mathbb{R}^{d\times k}$, due to elementary linear algebra we have, $||XX^TM||_F^2 \leq ||M||_F^2$.*

In the following, we state the definitions of subspace and projective clustering.

Definition 8 (Subspace clustering). Let $A \in \mathbb{R}^{n\times d}$ and j be an integer less than d. Then, the problem of j-subspace clustering is to find a j-dimensional subspace L of $\mathbb{R}^d$ that minimizes the $\text{dist}^2(A, L)$. In other words, the goal is to find a matrix $X^{\perp} \in \mathbb{R}^{d\times(d-j)}$ having orthonormal columns that minimizes $||AX^{\perp}||_F^2$ over every such possible matrix $X^{\perp}$.

Definition 9 (linear (affine) (k, j)-projective clustering). Let $A \in \mathbb{R}^{n\times d}$, j be an integer less than d, and k be an integer less than n. Then, the problem of linear (affine) (k, j)-projective clustering is to find a closed set $\mathcal{C}$, which is the union of k linear (affine) subspaces $\{L_1, \ldots L_k\}$ each of dimension at most j, such that it minimizes the $\text{dist}^2(A, \mathcal{C})$, over every possible choice of $\mathcal{C}$.

Theorem 2 (Low-rank approximation by [17]). *Let $A \in \mathbb{R}^{n\times d}$, and $\pi_{.}(.)$ denote the projection operators stated in the notation table. If $\epsilon \in (0, 1]$ and $\mathcal{S}$ is an $(r \times n)$ Johnson-Lindenstrauss matrix with i.i.d. zero-mean ± 1 entries and $r = O\left((\frac{m}{\epsilon} + m\log m)\log\frac{1}{\delta}\right)$, then with probability at least $1 - \delta$ it holds that*

$$||A - \pi_{\mathcal{S}A,m}(A)||_F^2 \leq (1 + \epsilon)||A - A^{(m)}||_F^2.$$

Further, computing the singular vectors spanning $\pi_{\mathcal{S}A,m}(A)$ in two passes[2] over the data requires $O(\mathbf{nnz}(A)r + (n + d)r^2)$ time.

For our analysis, we will use a weak triangle inequality which is stated below:

Lemma 10 (Lemma 7.1 of [10]). *For any $\varepsilon \in (0, 1)$, a closed set $\mathcal{C}$, and two points $p, q \in \mathbb{R}^d$, we have*

$$|\text{dist}^2(p,\mathcal{C}) - \text{dist}^2(q,\mathcal{C})| \leq \frac{12||p-q||^2}{\varepsilon} + \frac{\varepsilon}{2}\text{dist}^2(p,\mathcal{C}).$$

[2] Two passes are required as we first multiply A on the right with a Johnson-Lindenstrauss matrix $\mathcal{S}$, and then we project the rows of A again onto the row span of $\mathcal{S}A$.

3 Faster Coreset Construction for Subspace Clustering

In this section after revisiting the results of Cohen *et al.* [5], we present a randomized coreset construction for subspace clustering. The deterministic coreset construction of Feldman *et al.* [10] for subspace clustering problem relies on projecting the input matrix on its first few right singular vectors – projecting the rows of A on first few right singular vectors of A – which requires SVD computation of A. Cohen *et al.* [5] suggested that projecting the rows of A on some orthonormal vectors that closely approximate the right singular vectors of A (obtained via e.g. [17]) also satisfies the required properties of coreset *w.h.p.*, and as a consequence, gives a faster coreset construction.

Theorem 3 (Adapted from Theorem 8 of [5]). *Let $X \in \mathbb{R}^{d\times j}$ be an orthonormal matrix representing a subspace L, let $X^{\perp} \in \mathbb{R}^{d\times(d-j)}$ be the orthonormal matrix representing the orthogonal complement of L, $\epsilon \in (0,1)$, $\delta \in (0,1)$, $m = \lceil \frac{j}{\epsilon} \rceil$, $\Delta = ||A - A^{(m)}||_F^2$, and $\tilde{A}$ is a rank m approximation of A satisfying Theorem 2. Then, the following is true with probability at least $1-\delta$:*

$$0 \leq \left| ||\tilde{A}X^{\perp}||_F^2 + \Delta - ||AX^{\perp}||_F^2 \right| \leq 2\epsilon ||AX^{\perp}||_F^2.$$

Proof. Using a result of Sarlós [17], we get a rank m approximation of A. If $\mathcal{S}$ is an $(r \times n)$ JL matrix, where $r = O\left(\left(\frac{m}{\epsilon} + m\log m\right)\log\frac{1}{\delta}\right)$ (see Theorem 2) then the following is true with probability at least $1-\delta$:

$$||A - \pi_{\mathcal{S}A,m}(A)||_F^2 \leq (1+\epsilon)||A - A^{(m)}||_F^2. \tag{1}$$

Here, $A^{(m)}$ is the best m rank approximation of A. Let R' be the matrix having the first m right singular vectors of $\pi_{\mathcal{S}A}(A)$, and let we denote $AR'R'^T$ by $\tilde{A}$, then by Eq. 1, the following holds true with probability at least $1-\delta$:

$$||A - \tilde{A}||_F^2 \leq (1+\epsilon)||A - A^{(m)}||_F^2 \tag{2}$$

In the following we show an upper bound on the following expression:

$$\begin{aligned}
&\left| ||\tilde{A}X^{\perp}||_F^2 + \Delta - ||AX^{\perp}||_F^2 \right| \\
&= \left| ||\tilde{A}||_F^2 - ||\tilde{A}X||_F^2 + ||A - A^{(m)}||_F^2 - ||A||_F^2 + ||AX||_F^2 \right| && (3)\\
&= \left| ||\tilde{A}||_F^2 - ||\tilde{A}X||_F^2 + ||A||_F^2 - ||A^{(m)}||_F^2 - ||A||_F^2 + ||AX||_F^2 \right| && (4)\\
&= \left| ||\tilde{A}||_F^2 - ||A^{(m)}||_F^2 - ||\tilde{A}X||_F^2 + ||AX||_F^2 \right| \\
&\leq \left| ||A^{(m)}||_F^2 - ||A^{(m)}||_F^2 + ||AX||_F^2 - ||\tilde{A}X||_F^2 \right| && (5)\\
&= \left| ||AX||_F^2 - ||\tilde{A}X||_F^2 \right| \leq 2\epsilon ||AX^{\perp}||_F^2 && (6)
\end{aligned}$$

Equality 3 follows from Pythagorean theorem; Equality 4 follows from Fact 3, where $A^{(m)} = AV'V'^T$, and $V' \in \mathbb{R}^{d\times m}$ having m columns from the first m right

singular vectors of A; Inequality 5 holds as the value of $||\tilde{A}||_F^2 - ||A^{(m)}||_F^2$ can be at most zero, because at the best we can hope to sample the right singular vectors of A as R', which maximizes the value of the desired expression; finally Inequality 6 holds from Lemma 11.

A proof of the following lemma follows from the analysis of Lemma 5 of [5]. We defer it to the full version of this paper.

Lemma 11 (Adapted from Lemma 5 of [5]). *Let $A \in \mathbb{R}^{n\times d}$, $\tilde{A}$ is a rank m approximation of A satisfying Eq. 2, then*

$$0 \leq ||AX||_F^2 - ||\tilde{A}X||_F^2 \leq 2\epsilon||AX^{\perp}||_F^2.$$

4 Faster Coreset Construction for Projective Clustering

In this section, extending the result (Theorem 3) of the previous section, we present a randomized coreset construction for the problem of projective clustering. More precisely, if $L_1, ..., L_k$ be a set of k subspaces each of dimension at most j, and let $\mathcal{C}$ be a closed set containing union of them, then our randomized coreset is a matrix of very small rank (independent of d) and it approximately preserves the distances from every such closed set $\mathcal{C}$, with high probability. Our main contribution is the dimensionality reduction step of the coreset construction, which is presented in Algorithm 1 below.

1 **Input**: $A \in \mathbb{R}^{n\times d}$, an integer $1 \leq j < d-1$, and an integer $1 \leq k < n-1$ such that $j^* \leq d-1$, where $j^* = k(j+1)$, $\epsilon \in (0,1)$, $\delta \in (0,1)$.
2 **Result**: Dimensionality reduction for randomized coreset construction for the projective clustering.
3 Compute an Johnson-Lindenstrauss matrix $[\mathcal{S}]_{r\times n}$ having *i.i.d.* ± 1 entries and zero-mean, where $r = O((\frac{m^*}{\epsilon} + m^* \log m^*) \log \frac{1}{\delta})$, $m^* = \lceil \frac{52j^*}{\epsilon^2} \rceil$.
4 Compute the matrix $\pi_{\mathcal{S}A}(A)$.
5 Compute the SVD of $\pi_{\mathcal{S}A}(A)$, let $R^* \in \mathbb{R}^{d\times m^*}$ be the first m^* right singular vectors of $\pi_{\mathcal{S}A}(A)$.
6 Let us denote AR^*R^{*T} by A^*, and output A^*.

Algorithm 1. Dimensionality reduction for projective clustering.

Proof of Theorem 1: Let $[X^*]_{d\times j^*}$ be a matrix with orthonormal columns whose span is L^*, and let $L^{*\perp}$ be the orthogonal complement of L^* spanned by $[X^{*\perp}]_{d\times(d-j^*)}$. If $\mathcal{C}$ is a closed set spanned by L^*, then due to the Pythagorean theorem, we have, $\text{dist}^2(A,\mathcal{C}) = ||AX^{*\perp}||_F^2 + \text{dist}^2(AX^*X^{*T},\mathcal{C})$. Further,

$$\begin{aligned}
&\left|\left(\text{dist}^2(A^*,\mathcal{C}) + \Delta^*\right) - \text{dist}^2(A,\mathcal{C})\right| \\
&= \left|\left(||A^*X^{*\perp}||_F^2 + \text{dist}^2(A^*X^*X^{*T},\mathcal{C}) + \Delta^*\right) - \left(||AX^{*\perp}||_F^2 + \text{dist}^2(AX^*X^{*T},\mathcal{C})\right)\right| \\
&\leq \underbrace{\left|\left(||A^*X^{*\perp}||_F^2 + \Delta^* - ||AX^{*\perp}||_F^2\right)\right|}_{\text{first term}} + \underbrace{\left|\left(\text{dist}^2(A^*X^*X^{*T},\mathcal{C}) - \text{dist}^2(AX^*X^{*T},\mathcal{C}))\right)\right|}_{\text{second term}}
\end{aligned}$$

We have to bound two terms in the above expression. The first term can be upper bounded using a similar analysis as of Theorem 3 which holds true with probability at least $1-\delta$. (In Theorem 3, we replace j by j^*, m by m^*, ϵ by $\frac{\epsilon^2}{52}$, and Δ by Δ^*).

$$\left|||A^*X^{*\perp}||_F^2 + \Delta^* - ||AX^{*\perp}||_F^2\right| \leq \frac{\epsilon^2}{26}||AX^{*\perp}||_F^2 \tag{7}$$

To bound the second term $\left|\text{dist}^2(A^*X^*X^{*T}, \mathcal{C}) - \text{dist}^2(AX^*X^{*T}, \mathcal{C}))\right|$, we use Lemma 10. For any $\varepsilon \in (0,1)$ and due to Lemma 10, we have

$$\begin{aligned}
&\left|\text{dist}^2(A^*X^*X^{*T}, \mathcal{C}) - \text{dist}^2(AX^*X^{*T}, \mathcal{C}))\right| \\
&\leq \frac{12}{\varepsilon}||A^*X^*X^{*T} - AX^*X^{*T}||_F^2 + \frac{\varepsilon}{2}\text{dist}^2(AX^*X^{*T}, \mathcal{C}) \\
&\leq \frac{12}{\varepsilon}\left(\frac{\epsilon^2}{26}||AX^{*\perp}||_F^2\right) + \frac{\varepsilon}{2}\text{dist}^2(AX^*X^{*T}, \mathcal{C}) \qquad (8) \\
&\leq \frac{12}{\varepsilon}\left(\frac{\epsilon^2}{26}||AX^{*\perp}||_F^2\right) + \frac{\varepsilon}{2}\text{dist}^2(A, \mathcal{C})
\end{aligned}$$

Inequality 8 holds due to Lemma 12. Thus, we have

$$\left|\text{dist}^2(A^*X^*X^{*T}, \mathcal{C}) - \text{dist}^2(AX^*X^{*T}, \mathcal{C}))\right| \leq \frac{12}{\varepsilon}\left(\frac{\epsilon^2}{26}||AX^{*\perp}||_F^2\right) + \frac{\varepsilon}{2}\text{dist}^2(A, \mathcal{C}) \tag{9}$$

Equation 7, in conjunction with Eq. 9, gives us the following:

$$\begin{aligned}
&\left|\left(\text{dist}^2(A^*, \mathcal{C}) + \Delta^*\right) - \text{dist}^2(A, \mathcal{C})\right| \\
&\leq \left(1 + \frac{12}{\varepsilon}\right)\frac{\epsilon^2}{26}||AX^{*\perp}||_F^2 + \frac{\varepsilon}{2}\text{dist}^2(A, \mathcal{C}) \\
&\leq \left(1 + \frac{12}{\varepsilon}\right)\frac{\epsilon^2}{26}\text{dist}^2(A, \mathcal{C}) + \frac{\varepsilon}{2}\text{dist}^2(A, \mathcal{C}) \\
&= \left(\frac{\epsilon^2}{26} + \frac{12\epsilon^2}{26\varepsilon} + \frac{\varepsilon}{2}\right)\text{dist}^2(A, \mathcal{C}) \\
&= \left(\frac{\epsilon^2}{26} + \frac{12\epsilon}{26} + \frac{\epsilon}{2}\right)\text{dist}^2(A, \mathcal{C}) \qquad (10) \\
&\leq \epsilon\text{dist}^2(A, \mathcal{C})
\end{aligned}$$

Equality 10 holds by choosing $\varepsilon = \epsilon$, and as $\epsilon^2/26 + 12\epsilon/26 < \epsilon/2$.

A proof of the following lemma is deferred to the full version of this paper.

Lemma 12. *Let $X^* \in \mathbb{R}^{d\times j^*}$ be a matrix with orthonormal columns whose span is L^*, then in Algorithm 1 the following is true with probability at least $1-\delta$*

$$||A^*X^*X^{*T} - AX^*X^{*T}||_F^2 \leq \frac{\epsilon^2}{26}||AX^{*\perp}||_F^2.$$

Remark 4. Please note that it is sufficient to store the matrix AR^* which is of dimension m^*, where $m^* = O\left(k(j+1)/\epsilon^2\right)$. However, for the purpose of our analysis, we use the matrix AR^*R^{*T} which is of dimension d, and rank m^*. Further, the space that is required to store our coreset is $O(nm^*+1)$ – we need $O(nm^*)$ space to store the matrix AR^*, and $O(1)$ space to store the term Δ^*; on the other hand, the space require to store A is $O(nd)$.

Comparison with coreset construction of [10]: Coreset construction of [10] requires projecting the rows of A on its first $O(k(j+1)/\epsilon^2)$ right singular vectors which gives a matrix of rank $O(k(j+1)/\epsilon^2)$ and it approximately preserves the distance from any closed $\mathcal{C}$. Their construction requires computing SVD of the given matrix A, which has the run-time complexity of $\min\{n^2d, nd^2\}$. In our construction, we showed that it is also sufficient to project the rows of A on $O(k(j+1)/\epsilon^2)$ orthonormal vectors that closely approximate the right singular vectors of A. We now give an expected time bound on the running time of Algorithm 1. Time required for execution of line number $3, 4, 5$ is

$$O\left(\mathbf{nnz}(A)(\frac{m^*}{\epsilon}+m^*\log m^*)+(n+d)(\frac{m^*}{\epsilon}+m^*\log m^*)^2\right)$$
$$=O\left(\mathbf{nnz}(A)(\frac{j^*}{\epsilon^3}+\frac{j^*}{\epsilon^2}\log\frac{j^*}{\epsilon^2})+(n+d)(\frac{j^*}{\epsilon^3}+\frac{j^*}{\epsilon^2}\log\frac{j^*}{\epsilon^2})^2\right),$$

due to [17], where $j^* = k(j+1)$. Further, line number 6 requires time - for projecting A on R^*, which due to an elementary matrix multiplication is $O(ndm^*) = O\left(\frac{ndj^*}{\epsilon^2}\right)$. Thus, total expected running time of Algorithm 1 is

$$O\left(\mathbf{nnz}(A)\left(\frac{j^*}{\epsilon^3}+\frac{j^*}{\epsilon^2}\log\frac{j^*}{\epsilon^2}\right)+(n+d)\left(\frac{j^*}{\epsilon^3}+\frac{j^*}{\epsilon^2}\log\frac{j^*}{\epsilon^2}\right)^2+\frac{ndj^*}{\epsilon^2}\right)$$
$$=\tilde{O}\left(\mathbf{nnz}(A)\frac{j^*}{\epsilon^3}+(n+d)\frac{j^{*2}}{\epsilon^6}+\frac{ndj^*}{\epsilon^2}\right).$$

Clearly, if $n \geq d$ and $j^* = o(n)$, or, if $n < d$ and $j^* = o(d)$, then our expected running time is better than that of [10].

5 Conclusion and Open Problems

We presented a randomized coreset construction for projective clustering *via* low rank approximation. We first revisited the result of [5] for the subspace clustering, and then extended their result to construct a randomized coreset for projective clustering. We showed that our construction is significantly faster (when the values of k and j are small), as compared to the corresponding deterministic construction of [10], and it also maintains nearly the same accuracy. Our work leaves several open problems - improving the dimensionality reduction bounds for projective clustering, or giving a matching lower bound for the same. Another important open problem is to come up with the dimension reduction step of coreset construction using feature selection algorithms such as *row/column subset selection* [3].

References

1. Agarwal, P.K., Har-Peled, S., Varadarajan, K.R.: Approximating extent measures of points. J. ACM **51**(4), 606–635 (2004)
2. Agarwal, P.K., Har-Peled, S., Varadarajan, K.R.: Geometric approximation via coresets. In: Welzl, E., (ed.) Current Trends in Combinatorial and Computational Geometry (2007)
3. Boutsidis, C., Mahoney, M.W., Drineas, P.: An improved approximation algorithm for the column subset selection problem. In: Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, 4–6 January 2009, pp. 968–977 (2009)
4. Clarkson, K.L., Woodruff, D.P.: Low rank approximation and regression in input sparsity time. In: Symposium on Theory of Computing Conference, STOC 2013, Palo Alto, CA, USA, 1–4 June 2013, pp. 81–90 (2013)
5. Cohen, M.B., Elder, S., Musco, C., Musco, C., Persu, M.: Dimensionality reduction for k-means clustering and low rank approximation. In: Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, 14–17 June 2015, pp. 163–172 (2015)
6. Ester, M., Kriegel, H., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD 1996), Portland, Oregon, USA, pp. 226–231 (1996)
7. Feldman, D., Fiat, A., Sharir, M.: Coresets forweighted facilities and their applications. In: 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21–24 October 2006, Berkeley, California, USA, Proceedings, pp. 315–324 (2006)
8. Feldman, D., Langberg, M.: A unified framework for approximating and clustering data. In: Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6–8 June 2011, pp. 569–578 (2011)
9. Feldman, D., Monemizadeh, M., Sohler, C., Woodruff, D.P.: Coresets and sketches for high dimensional subspace approximation problems. In: Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 630–649 (2010)
10. Feldman, D., Schmidt, M., Sohler, C.: Turning big data into tiny data: Constant-size coresets for k-means, PCA and projective clustering. In: Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1434–1453 (2013)
11. Har-Peled, S.: No, coreset, no cry. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science. Lecture Notes in Computer Science, vol. 3328, pp. 324–335. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30538-5_27
12. Har-Peled, S., Mazumdar, S.: On coresets for k-means and k-median clustering. In: Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, 13–16 June 2004, pp. 291–300 (2004)
13. Henzinger, M.R., Raghavan, P., Rajagopalan, S.: Computing on data streams. In: Proceedings of a DIMACS Workshop on External Memory Algorithms, New Brunswick, New Jersey, USA, 20–22 May 1998, pp. 107–118 (1998)
14. Hinneburg, A., Aggarwal, C.C., Keim, D.A.: What is the nearest neighbor in high dimensional spaces? In: VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, 10–14 September 2000, Cairo, Egypt, pp. 506–515 (2000)

15. Mahoney, M.W.: Randomized algorithms for matrices and data. Found. Trends Mach. Learn. **3**(2), 123–224 (2011)
16. Phillips, J.M.: Coresets and sketches. CoRR, abs/1601.00617 (2016)
17. Sarlós, T.: Improved approximation algorithms for large matrices via random projections. In: 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21–24 October 2006, Berkeley, California, USA, Proceedings, pp. 143–152 (2006)
18. Varadarajan, K.R., Xiao, X.: A near-linear algorithm for projective clustering integer points. In: Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, 17–19 January 2012, pp. 1329–1342 (2012)
19. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: A new data clustering algorithm and its applications. Data Min. Knowl. Discov. **1**(2), 141–182 (1997)

Separating Interaction Effects Using Locating and Detecting Arrays

Stephen A. Seidel, Kaushik Sarkar, Charles J. Colbourn(✉), and Violet R. Syrotiuk

School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ, USA
{stephen.seidel,ksarkar1,colbourn,syrotiuk}@asu.edu

Abstract. The correctness and performance of complex engineered systems are often impacted by many factors, each of which has many possible levels. Performance can be affected not just by individual factor-level choices, but also by interactions among them. While covering arrays have been employed to produce combinatorial test suites in which every possible interaction of a specified number of factor levels arises in at least one test, in general they do not identify the specific interaction(s) that are significant. Locating and detecting arrays strengthen the requirements to permit the identification of a specified number of interactions of a specified size. Further, to cope with outliers or missing responses in data collected from real engineered systems, a further requirement of *separation* is introduced. In this paper, we examine two randomized methods for the construction of locating and detecting arrays, the first based on the Stein-Lovász-Johnson paradigm, and the second based on the Lovász Local Lemma. Each can be derandomized to yield efficient algorithms for construction, the first using a conditional expectation method, and the second using Moser-Tardos resampling. We apply these methods to produce upper bounds on sizes of locating and detecting arrays for various numbers of factors and levels, when one interaction of two factor levels is to be detected or located, for separation of up to four. We further compare the sizes obtained with those from more targeted (and more computationally intensive) heuristic methods.

1 Introduction

Complex engineered systems are critical, engineered, large-scale systems, such as transportation networks, power grids, and wireless communication systems. The correct operation of such systems often depends not just on the choices made for numerous parameters in their configuration, but also on interaction effects among these choices. Moreover, the performance of such a system can be dramatically affected by the choices and their interactions, even when the system is operating.

We examine a formal testing model. There are k factors $F_1, \ldots, F_k$. Each factor F_i has a set $S_i = \{v_{i1}, \ldots, v_{is_i}\}$ of s_i possible values (*levels*). A *test* is an

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 349–360, 2018.
https://doi.org/10.1007/978-3-319-94667-2_29

assignment, for each $i = 1, \ldots, k$, of a level from $v_{i1}, \ldots, v_{is_i}$ to F_i. The execution of a test yields a measurement of a *response*. When $\{i_1, \ldots, i_t\} \subseteq \{1, \ldots, k\}$ and $\sigma_{i_j} \in S_{i_j}$, the set $\{(i_j, \sigma_{i_j}) : 1 \leq j \leq t\}$ is a *t-way interaction*. (The interaction has *strength* t.) A test on k factors *covers* $\binom{k}{t}$ t-way interactions. A *test suite* is a collection of tests. Usually such a test suite is represented as an array: Suppose that $A = (\sigma_{i,j})$ is an $N \times k$ array for which $\sigma_{i,j} \in S_j$ when $1 \leq i \leq N$ and $1 \leq j \leq k$. This is a *test suite* of *size* N and *type* $(s_1, \ldots, s_k)$. Types can be written in exponential notation: $g_1^{u_1} \cdots g_r^{u_r}$ means that there are u_i factors with g_i levels for $1 \leq i \leq r$. Tests are recorded as rows of A, and factors correspond to columns.

Combinatorial testing [20,28] is concerned with the design and analysis of test suites in order to assess correctness and performance of a system. The focus has been on test suites known as covering arrays, which ensure that every t-way interaction is covered by a test. We define these precisely next. Let $A = (\sigma_{i,j})$ be a test suite of size N and type $(s_1, \ldots, s_k)$. Let $T = \{(i_j, \sigma_{i_j}) : 1 \leq j \leq t\}$ be a t-way interaction. Denote by $\rho(A, T)$ the set $\{r : a_{r,i_j} = \sigma_{i_j}, 1 \leq j \leq t\}$ of rows of A in which the interaction is covered. A *mixed covering array* $\mathrm{MCA}(N; t, (s_1, \ldots, s_k))$ is a test suite A for which every t-way interaction T has $\rho_A(T) \neq \emptyset$, i.e., every t-way interaction is covered in at least one row. When used for testing correctness, covering arrays reveal the presence of an interaction that causes faulty behaviour, but in general does not identify the specific faulty interaction(s); see [10,11].

We consider two motivating examples. In [1], a software simulation of a mobile wireless network is studied. There, 75 factors are identified among the controllable parameters in the protocol stack, ranging from 2 to 10 levels. (The type is $10^8 9^1 8^4 7^5 6^{10} 5^4 4^6 3^9 2^{28}$.) Throughput is measured as a response, and the objective is to determine which factors, and interactions among them, significantly affect the response. In this setting, it is of little interest to determine whether some interactions significantly affect the response; the goal is to ascertain which do. One could, of course, obtain the responses for a MCA, and design further testing based on the interactions covered in tests whose responses deviate most widely from the mean. In this way, a MCA could be used to narrow the set of t-way interactions that might have a significant effect on performance. Because the study employs a software simulation, a second round of testing could be conducted in the same environment as the first, and an adaptive method that uses results of earlier tests to choose later ones may be suitable.

In [12], a testbed evaluation of a conferencing scenario in a wireless network is conducted, measuring voice quality and exposure as responses. There are 24 controllable factors, ranging from 2 to 5 levels. (The type is $5^9 4^5 3^7 2^3$.) Hence the testing environment seems somewhat simpler than the simulation. Unlike the well-controlled environment in which the simulation study is conducted, however, the conferencing scenario is impacted by factors in the physical environment, including the hardware used directly in the experiment, interference from other communications in the vicinity, and the like. Despite best efforts to shield the testbed from such effects, measurements taken far apart in time can

be significantly affected by environmental factors that cannot be controlled, and may not be measured. In this setting, it is desirable that all tests be conducted in the same environment, and the significant interactions be identified without further testing. Hence we want a *nonadaptive* approach to testing.

The nonadaptive identification of faults or significant interactions can be accomplished by strengthening covering arrays. A combinatorial set of requirements was first identified in [10]; see also [11,23]. We develop this next.

1.1 Locating and Detecting Arrays

Let A be a test suite of size N and type $(s_1, \ldots, s_k)$. Let $\mathcal{I}_t$ be the set of all t-way interactions for A, and let $\overline{\mathcal{I}_t}$ be the set of all interactions of strength *at most* t. When interaction $T \in \overline{\mathcal{I}_t}$ has strength less than t and the t-way interaction T' contains T, it must hold that $\rho(A, T') \subseteq \rho(A, T)$; there can be no row covering T' but not T. A set $\mathcal{T}' \subseteq \overline{\mathcal{I}_t}$ is *independent* if there do not exist $T, T' \in \mathcal{T}'$ with $T \subset T'$. Our objective is to identify a set $\mathcal{T} \subseteq \mathcal{I}_t$ (or perhaps $\overline{\mathcal{I}_t}$) that have significant effects on the response. If no limitation is placed on $\mathcal{T}$, the design of a test suite can be impossible [23]; even when possible, the size of the test suite grows as the number of interactions in $\mathcal{T}$ increases [10]. We assume that a number d of interactions is to be identified. When at most d are to be identified, we employ the notation $\overline{d}$. Despite this limit, in the intended applications, often many more than d significant interactions can be found by iterative analysis of the response data, adjusting the responses after each selection of significant interactions, without the need for further experimentation; see [31] for details.

Locating arrays for identifying sets of interactions can be defined in this framework [10]. For a set $\mathcal{T}$ of interactions, define $\rho(A, \mathcal{T}) = \bigcup_{T \in \mathcal{T}} \rho(A, T)$. A test suite A is (d, t)-*locating* if $\rho(A, \mathcal{T}_1) = \rho(A, \mathcal{T}_2) \Leftrightarrow \mathcal{T}_1 = \mathcal{T}_2$ whenever $\mathcal{T}_1, \mathcal{T}_2 \subseteq \mathcal{I}_t$, $|\mathcal{T}_1| = d$, and $|\mathcal{T}_2| = d$. When $\mathcal{T}_1, \mathcal{T}_2 \subseteq \overline{\mathcal{I}_t}$, and $\mathcal{T}_1$ and $\mathcal{T}_2$ are independent, the array is $(d, \overline{t})$-*locating*. When instead $|\mathcal{T}_1| \leq d$ and $|\mathcal{T}_2| \leq d$, the array is $(\overline{d}, t)$-*locating* or $(\overline{d}, \overline{t})$-*locating*.

Using a locating array, knowing the tests that yield a significant deviation in response, there can be at most one set of (at most) d interactions of strength (at most) t covered in the same sets of tests that account for these deviations. By enumeration of all sets of d interactions of strength t, the location of the interactions causing the faults can be calculated from the outcomes. However, determining the interactions involved may require enumeration of sets of interactions. Determining the tests that exhibit a substantial deviation in response does not ensure that any interaction that is covered only within these tests must be significant. To deal with this, additional requirements are needed [10].

An array A is (d, t)-*detecting* if $\rho(A, T) \subseteq \rho(A, \mathcal{T}_1) \Leftrightarrow T \in \mathcal{T}_1$ whenever $\mathcal{T}_1 \subseteq \mathcal{I}_t$, $|\mathcal{T}_1| = d$, and $T \in \mathcal{I}_t \setminus \mathcal{T}_1$. When instead $\mathcal{T}_1 \subseteq \overline{\mathcal{I}_t}$, $T \in \overline{\mathcal{I}_t}$, and $\mathcal{T}_1 \cup \{T\}$ is independent, the array is $(d, \overline{t})$-*detecting*. For detecting arrays, we can also consider a set of at most d interactions, to obtain arrays that are $(\overline{d}, t)$-*detecting* or $(\overline{d}, \overline{t})$-*detecting*.

Detecting arrays underlie an efficient algorithm for the recovery of the set of significant interactions [10], but necessitate a larger number of tests.

This framework of eight variants of testing arrays for identifying significant interactions suggests many existence questions. However, relationships among them enable a unified examination. In [10], the relevant relationships are established (provided that $s_1 > 1$, $k \geq t$, and d is not larger than the number of possible interactions):

$$\begin{array}{ccccc}
(\overline{d},\overline{t})\text{-detecting} & \Rightarrow & (\overline{d},t)\text{-detecting} & & \\
\Updownarrow & & \Updownarrow & & \\
(d,\overline{t})\text{-detecting} & \Rightarrow & (d,t)\text{-detecting} & & \\
\Downarrow & & \Downarrow & & \\
(\overline{d},\overline{t})\text{-locating} & \Rightarrow & (\overline{d},t)\text{-locating} & \Rightarrow & (d-1,t)\text{-detecting} \\
\Downarrow & & \Downarrow & & \\
(d,\overline{t})\text{-locating} & \Rightarrow & (d,t)\text{-locating} & \Rightarrow & (d-1,t)\text{-locating}
\end{array}$$

Like covering arrays, locating and detecting arrays scale well to large numbers of factors. Indeed when the strength, number d of potentially significant interactions, and maximum number of levels are fixed, the number of tests required is $O(\log k)$ [10]. Unlike covering arrays, however, constructions for locating and detecting arrays have been much less studied. Although locating and detecting arrays are mixed covering arrays of strength t, the extension of covering array constructions requires substantial information about the tests in which interactions are covered. Naturally the objective is to employ as few tests as possible.

Only when $d = t = 1$ is the minimum number of tests in (d,t)-, $(\overline{d},t)$-, $(d,\overline{t})$-, and $(\overline{d},\overline{t})$-locating arrays known precisely [8]. The analogous situation for detecting arrays is explored in [21,24], and strong bounds are established that are exact infinitely often.

When $t \geq 2$, exact results are known for locating arrays when k is very small [32,34]; for larger numbers of factors, a small set of recursive constructions is available when $d = 1$ and $t = 2$ for locating arrays [7]. Beyond these few direct and recursive methods, computational methods have been developed for $(1,2)$-locating arrays [17,19,26] using constraint satisfaction techniques and one-row-at-a-time greedy methods. In order to address concerns with infeasible tests, Jin and Tsuchiya [17] extend the definition of locating arrays to account for testing constraints. These algorithmic methods do not treat detecting arrays; for locating arrays they limit the number of significant interactions, the strength, and the number of factors to small values. In our motivating problems, limiting the number and strength of interactions can be worthwhile, but techniques are needed to construct locating and detecting arrays for larger numbers of factors.

2 The Need for Separation

Consider the use of a locating or detecting array in an experimental setting. In principle, the responses for each test can identify the set of significant interactions whenever the assumptions on number and strengths of interactions are met. In practice, however, a problem arises. Suppose that two sets of (at most) d interactions, $\mathcal{T}_1$ and $\mathcal{T}_2$, have $|\rho(\mathcal{T}_1) \setminus \rho(\mathcal{T}_2)| = 1$. If this occurs, the response

measured in a single test is the sole 'witness' to the difference between the two. In the absence of noise or measurement error, one such witness suffices to differentiate. In our experiments, however, outliers and missing responses do occur. These compromise our ability to analyze the response data. This can be mitigated by exploring a number of possible sets of significant interactions, as in [31], rather than identifying a single set. As we have discussed, it cannot be effectively handled by simply running the test for the outlier or missing response again, without strong (and unjustified) assumptions about the stability of environmental factors.

Therefore we argue that effective locating and detecting arrays must allow for outliers and missing responses. Fortunately this can be treated by further requirements on the testing array, by enforcing a separation between sets of rows for different sets of interactions. We make this precise next. Let $\delta \geq 1$ be an integer, the *distance*. A test suite A is (d, t, δ)-*locating* if whenever $\mathcal{T}_1, \mathcal{T}_2 \subseteq \mathcal{I}_t$, $|\mathcal{T}_1| = d$, and $|\mathcal{T}_2| = d$, we have that

$$|(\rho(A, \mathcal{T}_1) \cup \rho(A, \mathcal{T}_2)) \setminus (\rho(A, \mathcal{T}_1) \cap \rho(A, \mathcal{T}_2))| < \delta \Leftrightarrow \mathcal{T}_1 = \mathcal{T}_2.$$

This requires that at least δ tests witness the difference. The variants for $\overline{d}$ and $\overline{t}$ are immediate.

Similarly, an array A is (d, t, δ)-*detecting* if whenever $\mathcal{T}_1 \subseteq \mathcal{I}_t$, $|\mathcal{T}_1| = d$, and $T \in \mathcal{I}_t \setminus \mathcal{T}_1$, we have that $T \in \mathcal{T}_1$ whenever $|\rho(A, T) \setminus \rho(A, \mathcal{T}_1)| < \delta$ or $|\rho(A, \mathcal{T}_1) \setminus \rho(A, T)| < \delta$. Again, definitions of the variants for $\overline{d}$ and $\overline{t}$ are straightforward.

Separation by distance δ ensures that any $\delta - 1$ or fewer tests can fail to provide a response, or provide an outlier response, without losing the differentiation supported by the locating or detecting array. In our motivating examples among many others, requiring larger distance for separation is desirable, but only if it can be accomplished without a dramatic increase in the number of tests.

The simplest technique to make a (d, t, δ)-locating array might be to concatenate the rows of δ (d, t)-locating arrays, or indeed to replicate each row of a single (d, t)-locating array, each δ times. This would enable the use of the few available methods for locating arrays, while increasing the separation as required. However, this appears to necessitate far too many tests.

3 Randomized and Derandomized Algorithms

We require algorithms to construct (d, t, δ)-locating arrays and (d, t, δ)-detecting arrays. The methods of most interest to us must not only handle a range of small values of d, t, and δ (each between, say, 1 and 4), but – more importantly – must handle reasonable numbers of factors (in the range of 50–100 at least). We do not expect to (or need to) produce the fewest tests possible, but naturally we prefer arrays with fewer tests.

Every $(\overline{1}, t)$-locating array is a mixed covering array of strength t, and hence the algorithmic paradigms that have been most effective for covering arrays appear to be natural candidates for construction of locating and detecting arrays. Among these, integer programming, constraint satisfaction, tabu search, and

simulated annealing have provided the best available upper bounds on the number of tests when the number of factors is small [6]. One-column-at-a-time [14,20] and one-row-at-a-time [5] greedy algorithms extend the range of numbers of factors treated, but do not outperform more sophisticated methods for few factors and small strength. Indeed for both methods, a post-optimization strategy [27] can often reduce the number of tests by investing more computation.

For larger numbers of factors and larger strengths yet, the best available upper bounds arise from randomized methods based on the Stein-Lovász-Johnson framework [18,22,33], and derandomized algorithms using conditional expectations [3,4]; and on the Lovász Local Lemma [2,13] with Moser-Tardos resampling [25] to yield both efficient construction techniques and the best asymptotic bounds [9,29,30].

We fully expect that computationally intensive methods and storage intensive methods can and will produce detecting and locating arrays with fewer tests than randomized and derandomized methods when the number of factors is relatively small and the search is tailored to specific choices of d, t, and δ (all 'small'); see, for example, [19]. In exploring randomized techniques, our objective is instead to develop algorithms that can be effectively used for a wide variety of construction problems, without undue time and storage requirements.

3.1 The Stein-Lovász-Johnson Framework and Conditional Expectation

Suppose that an $N \times k$ array A is specified. When A does not meet the requirements to be a locating or detecting array of the kind intended, it is because certain requirements are not met. For example, for (d, t, δ)-locating, when $\mathcal{T}_1 \neq \mathcal{T}_2$, $|\mathcal{T}_1| = d$, and $|\mathcal{T}_2| = d$, but $|(\rho(A, \mathcal{T}_1) \cup \rho(A, \mathcal{T}_2)) \setminus (\rho(A, \mathcal{T}_1) \cap \rho(A, \mathcal{T}_2))| = \mu < \delta$, the requirement is not met, and has *deficiency* $\delta - \mu$. (When a requirement is met, its deficiency is 0.) This notion of deficiency can be extended to requirements for other locating and detecting arrays in a straightforward manner. Then the *deficiency* of A is the sum of the deficiencies of all requirements.

When the deficiency of A is nonzero, a next test can be chosen to reduce the deficiency. Indeed if a test were chosen at random among all possible tests, the *expected reduction* in deficiency can be calculated. The Stein-Lovász-Johnson framework dictates that a next test be chosen to reduce the deficiency by at least this expectation. Choosing such a test at each stage ensures that no more tests are needed than in an entire array chosen at random whose expected deficiency is less than 1, and indeed the one-test-at-a-time method often employs much fewer tests.

An effective implementation of this strategy requires not only that the expected reduction in deficiency be calculated, but that a test be found to achieve this reduction. In [3,4], conditional expectation methods are used to demonstrate for covering arrays that the expectation can be efficiently calculated, and more importantly that the test needed can be constructed by choosing one entry at a time so as never to decrease the expectation. Although the details for locating and detecting arrays differ from the simpler requirements for covering arrays,

the strategy of [4] can be applied here as well. In the interests of space, we do not here repeat the details needed in order to explore the differences.

Every time a new test is to be added, deficiencies for each requirement until this point are needed. A storage intensive method can maintain this information as tests are added, but the number of pairs of sets of interactions can quickly exceed available storage even for arrays with a moderate number of factors. Instead a time-intensive version could recalculate the deficiency for each requirement when it is needed, incurring a substantial amount of recomputation. Recomputation may be feasible; however, the deficiency for a restriction may be recalculated many times in the construction of a single test.

3.2 The Lovász Local Lemma and Moser-Tardos Resampling

For covering arrays, asymptotic results based on the Lovász Local Lemma (such as [15,16,30]) improve upon those based on the Stein-Lovász-Johnson paradigm [3,5]. Despite this, the latter have typically yielded fewer tests in practice. Hence one might expect, for locating and detecting arrays, that the conditional expectation methods would be the preferred ones. As with covering arrays, however, the column resampling methods based on the Lovász Local Lemma again lead to methods that avoid the time/storage tradeoff incurred by the conditional expectation methods so they again provide viable construction algorithms, which we outline next.

Suppose that an $N \times k$ array A is specified. When A does not meet the requirements to be a locating or detecting array of the kind intended, some requirement has deficiency greater than 0. Following [25], we consider each requirement in an arbitrary but fixed order. If none is encountered that has nonzero deficiency, the array is the desired solution. Otherwise, the first time a requirement with nonzero deficiency is found, we identify all columns involved in all interactions of both sets, and randomly resample all of the entries in the same column. When this resampling occurs, checking is restarted at the first requirement and continues in the fixed order.

Moser and Tardos [25] show that the number of times resampling occurs is expected to be polynomial when the number of tests is that specified by the bound in the Lovász Local Lemma. As noted in [9], resampling can be applied when the number of tests is less than the bound, but in that case there is no guarantee that a solution can be found in a small expected number of resamplings (or indeed found at all). Nevertheless, resampling underlies a construction algorithm that typically reduces the number of tests below the bound.

To accomplish this, a random array is chosen with a number of tests equal to the bound. Column resampling proceeds as described until the array has deficiency 0. At this point, one test is removed, typically making the deficiency again nonzero, and a new round of column resampling is begun with the smaller number of tests.

In order to ensure termination, a threshold on the total number of column resamplings is set. Once this threshold is exceeded, the computation terminates

with the current number of tests. The deficiency of each requirement is recalculated every time this requirement is checked; no status information is stored for the requirements. Whereas the conditional expectation methods can employ the deficiency of an interaction up to k times for each test added, the column resampling methods limit this recalculation to the threshold. But the actual behaviour is much better than this worst case limit. Indeed, requirements that are later in the fixed order are checked only when all earlier requirements have deficiency 0. Hence although all requirements must be verified to ensure that the array is of the intended kind, typically a much smaller number is examined before we discover a requirement demonstrating that it is not. In practice, this results in a much smaller number of recalculations of deficiencies for requirements than one might have anticipated.

The method is flexible enough to permit construction for the variety of locating and detecting arrays described here, requiring less time and less storage than the conditional expectation methods (and less than any method that stores a status for each requirement). Naturally the question is whether such column resampling methods can yield useful test suites of an acceptable size. We address this in Sect. 4.

3.3 Local Optimization

Column resampling makes no explicit effort to reduce the deficiency, instead relying on the likelihood that a random replacement of the columns reduces the deficiency more than it increases it. When provided with an initial array of very low (but nonzero) deficiency having few tests, column resampling often increases the deficiency far more than it reduces it.

In order to study the effects of this, we examine a local optimization technique. At each iteration, we again consider each requirement in an arbitrary but fixed order. If none is encountered that has nonzero deficiency, the array is the desired solution. Otherwise, the first time a requirement with nonzero deficiency is found, we identify and randomly resample a column that is involved. If the resulting array has deficiency no larger than it did before, the new array replaces the old. Then no iteration increases the deficiency.

This shares the low storage footprint of the column resampling methods. The time invested is harder to compare. Although column resampling may make many resamplings that make the deficiency higher, each such resampling is triggered typically after few requirements are checked. In order to retain at least the progress made, local optimization checks every requirement at every iteration.

4 Some Computational Results

We implemented a (storage-intensive) conditional expectation method (Sect. 3.1), a column resampling method (Sect. 3.2), and a local optimization method (Sect. 3.3) for $(\overline{1}, \overline{2})$-locating arrays and for $(1, \overline{2})$-detecting arrays.

Our primary concerns are to (1) assess the effect of requiring larger separation on the numbers of tests required and (2) determine the feasibility of constructing locating and detecting arrays for scenarios with tens to hundreds of factors. In the discussion to follow, we repeatedly refer to Table 1.

The first column of Table 1 lists the types for which we applied one or more of the computational methods. These range from few factors (10) to a larger number (100). We select primarily types in which are numbers of levels are

Table 1. Numbers of tests in generated locating and detecting arrays. Testbed has type $5^9 4^5 3^7 2^3$; Simulation has type $10^8 9^1 8^4 7^5 6^{10} 5^4 4^6 3^9 2^{28}$.

Type	$(\overline{1}, \overline{2})$-locating					$(1, \overline{2})$-detecting				
	$\delta = 1$		$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta = 1$		$\delta = 2$	$\delta = 3$	$\delta = 4$
2^{10}	15	14	19	24	30	25	21	32	42	54
2^{15}	19	17	22	29	34	30	29	41	57	63
2^{20}	21	19	26	31	37	37	32	44	57	70
2^{50}	29	26	33	40	47	52	46	63	76	89
2^{75}	32	28	36	44	50	58	51	68		
2^{100}	34	31	39	46	53	62	57	74		
3^{10}	34	30	46	57	66	71	60	87	109	128
3^{15}	42	40	52	65	73	82	75	101	124	146
3^{20}	49	44	57	69	79	96	83	110	136	157
3^{50}	60	57	70	83	95	122	110	139	167	
3^{75}	66	62	76	90	103	135	121			
3^{100}	70	67	81	94	107	138	132			
4^{10}	71	65	86	104	122	135	118	161	201	235
4^{15}	78	76	96	116	133	153	139	185	219	
4^{20}	91	82	104	122	141	170	152	198		
4^{50}	113	106	129	148	168	217	196	244		
4^{75}	120	116	138	159	178	236	212			
4^{100}	127	123	146	166	187	248	236			
5^{10}	118	110	141	165	194	220	192	263	315	367
5^{15}	133	126	156	185	212	247	223	293		
5^{20}	150	138	169	198	225	275	243	313		
5^{50}	182	173	208	236		342	310	391		
5^{75}	198	189	223	256		367	337			
5^{100}	211	202	235	266		390	357			
$5^{10}2^{10}$	123	110	139	172	197	370	316	415	504	597
Testbed	117	114	144	169	194	313	266	367	450	533
Simulation	580	532	654			1883	1712			

equal, to observe the growth in number of tests as a function of the number of factors. We also report results for one mixed type, $5^{10}2^{10}$ to demonstrate how detecting arrays and locating arrays differ (more on this soon), and the two types from our motivating examples.

Sizes from the column resampling algorithm for separation distance 1 are reported in the first of the two columns under '$\delta = 1$', both for locating and detecting arrays, using a threshold of 1000 resamplings. As expected, the method runs relatively quickly, completing in less than a minute for type 3^{75}.

We employed the local optimization algorithm both to compare with column resampling for distance 1, and to extend to larger distances (up to $\delta = 4$ for many types). It proved to be somewhat more time consuming to complete 1000 iterations, for example taking approximately 6 min for the locating array of type 3^{75} with distance 1. Nevertheless, in all computations it yields a size smaller than that from column resampling.

Most interesting is the effect of increasing the separation distance. The sizes obtained suggest that one can do much better than replicating a solution with distance 1 δ times; in fact, for most of the types examined, enforcing distance 4 no more than doubles the number of tests used for distance 1 for locating arrays. This suggests that one can better cope with outliers and missing responses in experimentation using locating and detecting arrays, incurring a modest amount of additional testing.

Table 1 also illustrates substantial differences between locating arrays and detecting arrays. The efficient recovery algorithm for the latter appears to come at a high price. For types in which all factors have the same number of levels, the number of tests in a detecting array appears to be nearly twice the number for the corresponding locating array. However, for types in which factors have widely different numbers of levels, detection appears to cause a much larger increase; see the last three rows in Table 1. This is as one would anticipate, because when a factor has few levels, each level appears in many more tests on average. The likelihood that this larger set of tests contains all tests in which a much less frequently occurring 2-way interaction appears is consequently larger, necessitating a larger number of tests.

Finally we mention some results from the conditional expectation method. With our current implementation for detecting arrays with distance 1, storage and time limitations make it infeasible to handle large numbers of factors, so we report only a handful of results for few factors. For $k \in \{10, 15, 20\}$, while local optimization produces detecting arrays of type 3^k with 60, 75, and 83 tests, conditional expectation produces much smaller arrays of sizes 41, 48, and 54, respectively. Similar differences are found in other cases with few factors.

Nagamoto *et al.* [26] describe a greedy algorithm for $(1, 2)$-locating arrays of distance 1 that bears some resemblance to the conditional expectation method outlined here and apply it to a limited set of types with at most 20 factors. For $k \in \{10, 15, 20\}$, while local optimization produces locating arrays of type 5^k with 110, 136, and 148 tests, their greedy approach produces much smaller arrays of sizes 91, 105, and 113, respectively.

Evidently, column resampling and local optimization, at least within the number of iterations performed, produce numbers of tests that are far from minimum. Despite this, we have found them to yield acceptable results, better than expected in a randomly chosen array, for much larger numbers of factors than appear to be handled by existing methods.

Acknowledgements. This work is supported in part by the U.S. National Science Foundation grant #1421058, and in part by the Software Test & Analysis Techniques for Automated Software Test program by OPNAV N-84, U.S. Navy.

References

1. Aldaco, A.N., Colbourn, C.J., Syrotiuk, V.R.: Locating arrays: a new experimental design for screening complex engineered systems. SIGOPS Oper. Syst. Rev. **49**(1), 31–40 (2015)
2. Alon, N., Spencer, J.H.: The Probabilistic Method. Wiley-Interscience Series in Discrete Mathematics and Optimization, 3rd edn. Wiley, Hoboken (2008)
3. Bryce, R.C., Colbourn, C.J.: The density algorithm for pairwise interaction testing. Softw. Testing Verification Reliab. **17**, 159–182 (2007)
4. Bryce, R.C., Colbourn, C.J.: A density-based greedy algorithm for higher strength covering arrays. Softw. Testing Verification Reliab. **19**, 37–53 (2009)
5. Cohen, D.M., Dalal, S.R., Fredman, M.L., Patton, G.C.: The AETG system: an approach to testing based on combinatorial design. IEEE Trans. Softw. Eng. **23**, 437–444 (1997)
6. Colbourn, C.J.: Covering array tables: $2 \leq v \leq 25$, $2 \leq t \leq 6$, $t \leq k \leq 10000$ (2005–2017). www.public.asu.edu/~ccolbou/src/tabby
7. Colbourn, C.J., Fan, B.: Locating one pairwise interaction: three recursive constructions. J. Algebra Comb. Discrete Struct. Appl. **3**, 125–134 (2016)
8. Colbourn, C.J., Fan, B., Horsley, D.: Disjoint spread systems and fault location. SIAM J. Discrete Math. **30**, 2011–2016 (2016)
9. Colbourn, C.J., Lanus, E., Sarkar, K.: Asymptotic and constructive methods for covering perfect hash families and covering arrays. Des. Codes Crypt. **86**, 907–937 (2018)
10. Colbourn, C.J., McClary, D.W.: Locating and detecting arrays for interaction faults. J. Comb. Optim. **15**, 17–48 (2008)
11. Colbourn, C.J., Syrotiuk, V.R.: Coverage, location, detection, and measurement. In: 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 19–25. IEEE Press (2016)
12. Compton, R., Mehari, M.T., Colbourn, C.J., De Poorter, E., Syrotiuk, V.R.: Screening interacting factors in a wireless network testbed using locating arrays. In: IEEE INFOCOM International Workshop on Computer and Networking Experimental Research Using Testbeds (CNERT) (2016)
13. Erdős, P., Lovász, L.: Problems and results on 3-chromatic hypergraphs and some related questions. In: Infinite and Finite Sets, Colloq., Keszthely, vol. 2, pp. 609–627 (1973). Colloq. Math. Soc. János Bolyai, vol. 10, North-Holland, Amsterdam (1975)
14. Forbes, M., Lawrence, J., Lei, Y., Kacker, R.N., Kuhn, D.R.: Refining the in-parameter-order strategy for constructing covering arrays. J. Res. Nat. Inst. Stand. Tech. **113**, 287–297 (2008)

15. Francetić, N., Stevens, B.: Asymptotic size of covering arrays: an application of entropy compression. J. Combin. Des. **25**, 243–257 (2017)
16. Godbole, A.P., Skipper, D.E., Sunley, R.A.: t-covering arrays: upper bounds and Poisson approximations. Comb. Probab. Comput. **5**, 105–118 (1996)
17. Jin, H., Tsuchiya, T.: Constrained locating arrays for combinatorial interaction testing. CoRR abs/1801.06041 (2018). http://arxiv.org/abs/1801.06041
18. Johnson, D.S.: Approximation algorithms for combinatorial problems. J. Comput. Syst. Sci. **9**, 256–278 (1974)
19. Konishi, T., Kojima, H., Nakagawa, H., Tsuchiya, T.: Finding minimum locating arrays using a SAT solver. In: 2017 IEEE International Conference on Software Testing, Verification and Validation Workshops, ICST Workshops 2017, Tokyo, Japan, 13–17 March 2017, pp. 276–277 (2017)
20. Kuhn, D.R., Kacker, R., Lei, Y.: Introduction to Combinatorial Testing. CRC Press, Boca Raton (2013)
21. Li, P.C., Meagher, K.: Sperner partition systems. J. Combin. Des. **21**(7), 267–279 (2013)
22. Lovász, L.: On the ratio of optimal integral and fractional covers. Discrete Math. **13**(4), 383–390 (1975)
23. Martínez, C., Moura, L., Panario, D., Stevens, B.: Locating errors using ELAs, covering arrays, and adaptive testing algorithms. SIAM J. Discrete Math. **23**, 1776–1799 (2009/2010)
24. Meagher, K., Moura, L., Stevens, B.: A Sperner-type theorem for set-partition systems. Electron. J. Combin. **12**, Note 20, 6 (2005). (Electronic)
25. Moser, R.A., Tardos, G.: A constructive proof of the general Lovász local lemma. J. ACM **57**(2), Article no. 11, 15 (2010)
26. Nagamoto, T., Kojima, H., Nakagawa, H., Tsuchiya, T.: Locating a faulty interaction in pair-wise testing. In: 20th IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2014, Singapore, 18–21 November 2014, pp. 155–156 (2014)
27. Nayeri, P., Colbourn, C.J., Konjevod, G.: Randomized postoptimization of covering arrays. Eur. J. Comb. **34**, 91–103 (2013)
28. Nie, C., Leung, H.: A survey of combinatorial testing. ACM Comput. Surv. **43**(2), #11 (2011)
29. Sarkar, K., Colbourn, C.J.: Two-stage algorithms for covering array construction. CoRR abs/1606.06730 (2016). http://arxiv.org/abs/1606.06730
30. Sarkar, K., Colbourn, C.J.: Upper bounds on the size of covering arrays. SIAM J. Discrete Math. **31**, 1277–1293 (2017)
31. Seidel, S.A., Mehari, M.T., Colbourn, C.J., De Poorter, E., Moerman, I., Syrotiuk, V.R.: Analysis of large-scale experimental data from wireless networks. In: IEEE INFOCOM International Workshop on Computer and Networking Experimental Research Using Testbeds (CNERT) (2018)
32. Shi, C., Tang, Y., Yin, J.: Optimal locating arrays for at most two faults. Sci. China Math. **55**(1), 197–206 (2012)
33. Stein, S.K.: Two combinatorial covering theorems. J. Comb. Theory Ser. A **16**, 391–397 (1974)
34. Tang, Y., Colbourn, C.J., Yin, J.: Optimality and constructions of locating arrays. J. Stat. Theory Pract. **6**(1), 20–29 (2012)

An Efficient Representation of Partitions of Integers

Kentaro Sumigawa(✉) and Kunihiko Sadakane

Department of Mathematical Informatics, Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, Japan
{kentaro_sumigawa,sada}@mist.i.u-tokyo.ac.jp

Abstract. We introduce a data structure for representing a partition of an integer n, which uses $O(\sqrt{n})$ bits of space. This is constant multiple of the information theoretic lower bound. Three types of operations $\mathsf{access_p}, \mathsf{bound_p}, \mathsf{prefixsum_p}$ are supported in constant time by using the notion of conjugate of a partition. In order to construct this data structure, we also construct a data structure for representing a monotonic sequence, which supports the same operations in constant time and uses $O(\min\{\frac{1}{\delta}u\left(\frac{n}{u}\right)^{\delta}, \frac{1}{\delta}n\left(\frac{u}{n}\right)^{\delta}\})$ bits of space for any positive constant δ. (n is the number of terms, and u denotes the size of the universe.)

1 Introduction

A partition of an integer n represents ways of division of n objects, which is equivalent to Young diagrams and Ferrers diagrams. This is usually represented by a positive integer sequence $\{A\} = \langle A[0], A[1], A[2], \ldots\rangle$ which satisfies two conditions: (i) $\{A\}$ is monotonic (not necessarily strictly) decreasing sequence, (ii) the sum of all terms over $\{A\}$ is n. It is known that they are useful in study of group representation theory [4]. This notion is also used for the situation that objects are divided into some groups such as a representation of a permutation [7]. We consider a data structure which represents a partition of an integer and supports the following fundamental three operations:

– $\mathsf{access_p}(i) = A[i]$, $\mathsf{bound_p}(i) = \#\{j \mid A[j] \geq i\}$, $\mathsf{prefixsum_p}(i) = \sum_{j=0}^{i} A[j]$

Regarding a partition as a Young diagram, the operations $\mathsf{access_p}(i)$ and $\mathsf{bound_p}(i)$ represents the number of cells in i-th row and column, respectively. It takes $O(n \lg n)$ bits to store all the answers, which takes too much space, since the information theoretic lower bound is shown to be $3.7\sqrt{n} - \lg n + O(1)$ bits[1] (see Theorem 4 below).

Few studies have focused on space efficient representations of a partition of an integer and the operations on it. [2] showed that reordering the terms according to a certain order (which means they resign the condition (i)), some operations

This work was supported by JST CREST Grant Number JPMJCR1402, Japan.

[1] In this paper, $\lg x$ denotes $\log_2 x$.

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 361–373, 2018.
https://doi.org/10.1007/978-3-319-94667-2_30

can be supported in constant time with an $O(\sqrt{n})$ bit space data structure. In order to apply [2] to our problem, we have to store the correspondence between the order of the terms sorted in different ways, which could take $O(n \lg n)$ bits.

Though we can apply existing data structures for monotonic sequences to represent a partition, their space requirement is much larger than $3.7\sqrt{n}$ bits because condition (ii) is not taken into account. Let u be the size of the universe of a sequence and n be the number of terms. Thus we consider the sequences such that $\{a\} : \ 0 \leq a[0] \leq a[1] \leq \cdots \leq a[n-1] < u$. For access, when $u > n$, Elias-fano sequence [1,3] takes $n(2+\lg \frac{u}{n})$ bits and support the operation in constant time. In the case $u \simeq n$, this sequence can be encoded by a bit vector and we can operate these operations in constant time using the data structure called FID [9]. When $u = O(\frac{n}{\lg n})$, access can be done in constant time using a bit vector in [5] which uses $O(u(1 + \lg \frac{n}{u \lg n}))$ bits. However, under the condition of $u \ll n$, such as $n = u^c$ for some constant $c(> 1)$, few data structures can deal with. For example, [8] supports access in $O(\lg \lg u)$ time while bound operation is in constant time. FID takes $O(n \frac{\lg \lg n}{\lg n})$ bits of space and it is quite larger than the information theoretic lower bound $O(u \lg \frac{n}{u})$. For prefixsum, without the restriction of monotonicity of sequences, it can be represented as a bit vector [9]. However, using this method it takes $O(n \lg u)$ bits even if the bit vector is compressed into the information theoretic lower bound.

In this paper we propose a new method which can deal with the conditions (i) and (ii) at the same time effectively, which is difficult for existing methods. This method is based on the fact that most of short sequences, which are obtained by dividing the original sequence, are of equal values because of the condition (ii). This means we do not have to store the greater part of the original sequence. As a result we obtain the following Theorem.

Theorem 1. *Let $\{A\}$ be a partition of an integer n. There exists a data structure which uses $O(\sqrt{n})$ bits and supports* $\mathsf{access}_\mathsf{p}$, $\mathsf{bound}_\mathsf{p}$, *and* $\mathsf{prefixsum}_\mathsf{p}$ *in constant time.*

In order to prove Theorem 1, we consider a new data structure for increasing (instead of decreasing in order to simplify) monotonic sequence which supports operations access, bound, prefixsum in constant time (they are distinguished from $\mathsf{access}_\mathsf{p}$ etc. since access supports arbitrary access over any monotonic sequence without condition (ii)). We obtain following (some words which appear in theorems are defined in Sect. 2):

Theorem 2. *Given a monotonic increasing sequence $\{a\}$ which satisfies $0 \leq a[i] < u$ $(0 \leq i < n)$, there exists a data structure for* access *and* bound *for $\{a\}$ and its conjugate sequence in $O(\delta)$ time and* prefixsum *in $O(\delta^2)$ time which requires $O(\min\{\frac{1}{\delta} n^\delta u^{1-\delta}, \frac{1}{\delta} u^\delta n^{1-\delta}\})$ bits of space for any $\delta > 0$.*

Theorem 3. *Given a bit vector B of length l with b 1s, there exists a data structure for* access, rank, *and* select *on B which requires $O(\delta)$ time and $O(\min\{\frac{1}{\delta} l^\delta b^{1-\delta}, \frac{1}{\delta} l^\delta (l-b)^{1-\delta}\})$ bits of space for any $\delta > 0$.*

Compared to other data structures (see Table 1), our data structure uses less space than others if $l = b^c$ for some constant $c > 1$.

Table 1. Results for the size and the valid sparsity of data structure which support all operations access, rank_0, rank_1, select_0, select_1 in constant time over a bit vector of length l with b 1s.

Data structure	Space	Valid sparsity
[9]	$\lg \binom{l}{b} + \mathrm{O}\left(l \frac{\lg \lg l}{\lg l}\right)$	$b = \Omega\left(\frac{l}{\lg l}\right)$
[9]	$\mathrm{O}\left(l \frac{\lg \lg l}{\lg l}\right)$	$b = \mathrm{o}\left(\frac{l}{\lg l}\right)$
[5]	$\mathrm{O}\left(b(1 + \lg \frac{l}{b \lg l})\right)$	$\Theta\left(\frac{l}{(\lg l)^c}\right) \leq b \leq \Theta\left(\frac{l}{\lg l}\right)$
Ours	$\mathrm{O}(l^\delta b^{1-\delta})$	Any

2 Preliminaries

In this paper, we use zero-based indexing.

Let $IMS(n, u)$ be the set of monotonic increasing sequences with n terms and its range is $[0, u)$. Similarly, let $DMS(n, u)$ be the set of monotonic decreasing sequences with n terms and its range is $[0, u)$.

2.1 Conjugate of a Monotonic Sequence

We define the conjugate of a monotonic sequence.

Definition 1. *Let $\{a\}$ be a sequence which consists of n terms and its range is $[0, u)$. The sequence $\{\overline{a}\}$ defined as follows is called $\{a\}$'s conjugate sequence*

$$\overline{a}[i] \equiv \#\{j \mid a[j] > i\} \, (= \mathsf{bound}(\{a\}, i)) \quad (0 \leq i < u - 1).$$

Fact 1. *$\{\overline{a}\}$ belongs to $DMS(u - 1, n + 1)$.*

Example 1. Let $\{a\}$ be monotonic decreasing sequence $\langle 6, 5, 5, 4, 3, 1, 1, 1, 0, 0\rangle \in DMS(10, 8)$. Its conjugate is $\{\overline{a}\} = \langle 8, 5, 5, 4, 3, 1, 0\rangle \in DMS(7, 11)$

Lemma 1. *Given $\{a\} \in DMS(n, u)$, $a[i] = \overline{\overline{a}}[i]$ holds for every $0 \leq i < n$.*

Proof. By the definition of conjugate, we have

$$\begin{aligned} \overline{\overline{a}}[i] &= \#\{j \mid \overline{a}[j] > i\} \\ &= \#\{j \mid \#\{k \mid a[k] > j\} > i\}. \end{aligned}$$

Here $\{a\}$ is a monotonic decreasing sequence, $\{j \mid \#\{k \mid a[k] > j\} > i\} = \{0, 1, \ldots, a[i] - 1\}$ and $\overline{\overline{a}}[i] = a[i]$ holds. □

Corollary 1. *For any sequence $\{a\} \in IMS(n, u)$, $a[i] = \overline{\overline{a}}[n - i - 1]$ holds for every $0 \leq i < n$.*

Now we consider data structures storing a monotonic increasing sequence $\{a\} \in IMS(n, u)$, supporting constant time operations which access arbitrary terms of sequences $\{a\}$ and $\{\overline{a}\}$. We are going to show that the operations on a bit vector can be applied to the representation of monotonic sequences. Three operations access, rank_c, select_c on bit vector B are defined as follows:

- $\mathsf{access}(x, B)$: the x-th bit of B.
- $\mathsf{rank}_c(x, B)$: $\#\{i \mid i \leq x \text{ and } B[i] = c\}$.
- $\mathsf{select}_c(x, B)$: $\min\{i \mid \mathsf{rank}_c(i, B) = x\}$.

Lemma 2 (FID [9]). *Given a bit vector B of length l with b 1s, the operations $\mathsf{access}(x, B)$, $\mathsf{rank}_c(x, B)$ and $\mathsf{select}_c(x, B)$ can be done in constant time with a data structure of $\lg \binom{l}{b} + \mathrm{O}(l \frac{\lg l}{\lg \lg l})$ bits.*

Lemma 3. *Given a monotonic increasing sequence $\{a\} \in IMS(n, u)$, there exists a data structure for reporting an arbitrary term of $\{a\}$ and $\{\overline{a}\}$ in constant time which requires $n + u + \mathrm{o}(n + u)$ bits of space.*

Proof. Let B a bit vector of length at most $n + u - 1$ which is obtained by appending $a_i - a_{i-1}$ 0s and one 1 for $i = 0, 1, \ldots, n - 1$ (for convenience, we define $a_{-1} = 0$). Then B consists of n 1s and at most $u - 1$ 0s. Then, operations are supported in constant time as there exist the relations $a[i] = \mathsf{rank}_0(\mathsf{select}_1(i, B), B)$, $\overline{a}[i] = n - \mathsf{rank}_1(\mathsf{select}_0(i, B), B)$. □

2.2 Partitions of Integers

Definition 2. *When a sequence $\{A\}$ satisfies the following conditions,*

$$\sum_{i=0}^{k-1} A[i] = n,$$
$$A[0] \geq A[1] \geq \cdots \geq A[k-1] \,(\geq 1)\,,$$

$\{A\}$ is called a partition of an integer n.

Fact 2. *The conjugate of a partition of n is also a partition of n.*

The number of the possible partitions of n is called partition number p_n. The asymptotic expression of p_n is given by $\frac{1}{4\sqrt{3}n} \exp\left(\pi\sqrt{\frac{2n}{3}}\right)$ [6]. Therefore, the following Theorem holds.

Theorem 4. *The information theoretic lower bound for the number of bits of a representation needed to distinguish all the partition of n is $\pi\sqrt{\frac{2}{3}} \lg e\sqrt{n} - \lg n + \mathrm{O}(1)$ bits.*

3 Data Structures to Represent Monotonic Sequences

Before we consider partitions of integers, we construct a data structure for monotonic sequences. First, we are going to prove Theorem 2.

In order to reduce the space complexity, we "divide" an original sequence into multiple short sequences.

Lemma 4. *Any sequence* $\{a\}$ *which belongs to* $IMS(n, u)$ $(n \geq u)$ *can be represented by two sequences of* $IMS(\sqrt{nu}, u)$ *and three bit vectors of length at most* $2u$ *each, so that* access *on* $\{a\}$ *and* $\{\overline{a}\}$ *can be done in* $2T + \mathrm{O}(1)$ *time, where* T *is the time complexity for* access *on* $IMS(\sqrt{nu}, u)$.

Proof. Divide n terms into $\sqrt{nu}$ blocks of $d = \sqrt{n/u}$ terms. Let B_i denote the i-th block and $h[i] = a[d(i+1)] - a[di]$ denote the increment of the sequence in the block B_i, for $0 \leq i < \sqrt{nu}$. We call B_i *empty block* when $h[i] = 0$, *non-empty block* otherwise. Let $z(< u)$ be the number of non-empty blocks.

The sequence $\{a\}$ can be represented by the following sequences:

1. $\{a'\} \in IMS(\sqrt{nu}, u)$ satisfies $a'[j] = a[jd]$ $(0 \leq j < \sqrt{nu})$.
2. $\{a''\} \in IMS(zd, u)$, concatenating all non-empty blocks of the original sequence. Because the number of non-empty blocks is z, this sequence consists of at most $zd \leq \sqrt{nu}$ terms.

In addition, we construct three sequences:

3. $\{v\} \in IMS(u, u), v[k] = \#\{j \mid B_j \text{ is non-empty and } a''[j] < k\}$.
4. $\{s\} \in IMS(z, u), s[k] = (k\text{-th non-empty block's first term})$.[2]
5. $\{t\} \in IMS(z, u), t[k] = (k\text{-th non-empty block's last term})$.

Using Lemma 3, each sequence can be represented in at most $2u + \mathrm{o}(u)$ bits. Algorithms 1 and 2 show the procedures of operations, and Fig. 1 shows a visual image of Algorithm 2. □

Lemma 5. *Let* $\{a\} \in IMS(n, u)$ $(n \geq u)$ *be the original sequence. It can be represented in* $\mathrm{O}(2^k n^{1/2^k} u^{1-1/2^k})$ *bits of space in total for any fixed* $k > 0$. *The operation* access *over* $\{a\}$ *and* $\{\overline{a}\}$ *can be done in* $\mathrm{O}(2^k)$ *time. We denote this data structure* D_k.

Proof. Applying Lemma 4 k times recursively, the original sequence is decomposed into sequences and bit vectors without information loss. We can represent these sequences by a full binary tree of height k. The root is numbered 0, and the node i has two children $2i + 1$ and $2i + 2$. The root node corresponds to the original sequence $\{a\}$, and its two children correspond to the two sequences $\{a'\}$ and $\{a''\}$ obtained from $\{a\}$ using Lemma 4. Internal nodes store the bit vectors $\{v\}$, $\{s\}$ and $\{t\}$, while leaves store the sequences $\{a'\}$ and $\{a''\}$ using Lemma 3.

[2] We use this sequence in Sect. 4.

Space Complexity: Let $\{a^{(0)}\}$ be the original sequence. For integer i, when the sequence $\{a^{(i)}\}$ is represented by two sequences $\{a'\}$ and $\{a''\}$ as Lemma 4, we name them $\{a^{(2i+1)}\}$ and $\{a^{(2i+2)}\}$ respectively. Applying this operation for $0 \leq i \leq 2^k - 2$, we obtain 2^k sequences, each of which belong to $IMS(n^{1/2^k}u^{1-1/2^k}, u)$. Using Lemma 3, each sequence can be represented in $\mathrm{O}(n^{1/2^k}u^{1-1/2^k})$ bits of space. In addition, $3 \cdot 2^k$ bit vectors $\{v^{(i)}\}, \{s^{(i)}\}, \{t^{(i)}\}$ take $\mathrm{O}(2^k u)$ bits in total. This means that the whole space complexity is $\mathrm{O}(2^k n^{1/2^k} u^{1-1/2^k}) + \mathrm{O}(2^k u) = \mathrm{O}(2^k n^{1/2^k} u^{1-1/2^k})$.

Time Complexity: Let $T_1(k)$ be the time complexity for random access on $\{\overline{a}\}$ using D_k. From Algorithm 2[3], $T_1(k) = 2T_1(k-1) + \mathrm{O}(1)$ and $T_1(1) = \mathrm{O}(1)$ holds. Then, $T_1(k)$ is shown to be $\mathrm{O}(2^k)$.

Let $T_2(k)$ be the time complexity for random access on $\{a\}$ using D_k. Considering the function access which returns two values, $a[i-1]$ and $a[i]$, the time complexity is $T_2(k) = 2T_2(k-1) + \mathrm{O}(1)$ which means $T_2(k) = \mathrm{O}(2^k)$, as shown in Algorithm 1. □

Algorithm 1. access$(\{a\}, j) = (a[j], a[j+1])$ (if $\{a\}$ is assigned to an internal node)

```
d ← √(n/u), b ← j/d, r ← j%d
(p1, p2) ← access2({a'}, b)
if r ≠ √(nu) − 1 then
  if p1 ≠ p2 then
    return access2({a''}, v[p1] · √(nu) + r)
  else
    return (p2, p2)
  end if
else
  if p1 ≠ p2 then
    return (t[v[p1]], p2)
  else
    return (p1, p2)
  end if
end if
```

Lemma 5 shows that the operations access and bound are supported for the sequences such that $u \leq n$ since bound$(\{a\}, i)$=access$(\{\overline{a}\}, i)$. In the case that the sequence $\{a\}$ belongs to $IMS(n, u)$ such that $u > n$, applying Lemma 5 to its conjugate sequence, access and bound over $\{a\}$ is supported. Consequently, we obtain the first part of Theorem 2.

To prove Theorem 3, we convert a bit vector to monotonic sequences using the following lemma:

[3] In this paper, a/b and $a\%b$ appeared in pseudo codes means $\lfloor a/b \rfloor$ and a mod b, respectively.

Algorithm 2. $\mathsf{access}(\{\overline{a}\}, i)$ (if $\{a\}$ is assigned to an internal node)

if $a[0] > i$ **then**
  return n
end if
$d \leftarrow \sqrt{n/u}$, $l \leftarrow d \cdot \mathsf{access}(\{\overline{a'}\}, i)$, $w \leftarrow \mathsf{access}(\{\overline{a''}\}, i) \% d$
return $w + l$

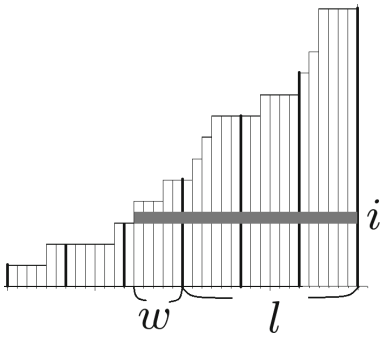


Fig. 1. The histogram representation of a monotonic increasing sequence (i-th bin's height is $a[i]$). The length of gray bar represents $\mathsf{access}(\{\overline{a}\}, i)$.

Lemma 6. *Let B be a bit vector of length l with b 1s. Suppose there exists a data structure for accessing an arbitrary term of sequence $IMS(n, u)$ and its conjugate sequence in $g(n, u)$ time which requires $f(n, u)$ bits of space, there exists a data structure for* access, rank_c, *and* select_c *on B in $\mathrm{O}(g(b, l) + g(l - b, b + 1))$ time which requires $f(b, l) + f(l - b, b + 1)$ bits of space.*

Proof. We construct two auxiliary sequences $\{s\} \in IMS(b, l)$ and $\{t\} \in IMS(l - b, b + 1)$, defined as follows:

$$s[i] = \mathsf{select}_1(i, B),$$
$$t[i] = \#\{j \leq \mathsf{select}_0(i, B) \mid B[j] = 1\} = \mathsf{rank}_1(\mathsf{select}_0(i, B)).$$

Each auxiliary sequence is represented in $f(b, l)$ bits and $f(l - b, b + 1)$ bits of space, respectively.

Using these sequences, the following operations are supported in $\mathrm{O}(g(b, l) + g(l - b, b + 1))$: (for convenience, we define $\overline{s}[-1] = n$ and $\overline{s}[n - 1] = 0$.)

- $\mathsf{access}(x, B)$: if $\exists i\ s[i] = x (\Leftrightarrow \overline{s}[x - 1] > \overline{s}[x])$ return 1, otherwise return 0.
- $\mathsf{select}_1(x, B)$: $s[x]$.
- $\mathsf{select}_0(x, B)$: $x + t[x]$.
- $\mathsf{rank}_1(x, B)$: $\#\{j \mid s[j] \leq x\} = n - \#\{j \mid s[j] > x\} = n - \overline{s}[x]$.
- $\mathsf{rank}_0(x, B)$: $x - \mathsf{rank}_1(x) = x - n + \overline{s}[x]$. □

From Theorem 2 and Lemma 6, we obtain Theorem 3.

4 A Data Structure for Prefix Sums

In this section we consider prefix sums over monotonic sequences. The prefix sum over $IMS(n,u)$ takes a value in the range $[0,(u-1)n]$. Thus, regarding the prefix sums as $IMS(n,(u-1)n+1)$, it takes $\mathrm{O}(n \lg u)$ bits of space to store all the values. We develop a data structure with the same space complexity as Lemma 5 which is obviously smaller than $\mathrm{O}(n \lg u)$ bits of space using Lemma 2.

Lemma 7. *Given a monotonic increasing sequence $\{a\} \in IMS(n,u)$, there exists a data structure for reporting arbitrary prefix sum of $\{a\}$ and $\{\overline{a}\}$ in constant time which requires $\mathrm{O}(n+u)$ bits of space.*

Proof. Let B be a bit vector of length $l(=n+u)$ which represents $\{a\}$ by using Lemma 3. The k-th prefix sum $\sum_{i=0}^{k-1} a[i]$ is equal to $\sum_{i=0}^{k-1} \mathsf{rank}_0(\mathsf{select}_1(i))$. We divide B into $\frac{2l}{\lg l}$ blocks of length $d = \frac{\lg l}{2}$. The k-th block stores a value s_k explicitly where s_k is defined as follows:

$$s_k = \sum_{i=0}^{\mathsf{rank}_1(kd-1,B)} \mathsf{rank}_0(\mathsf{select}_1(i,B),B).$$

Since the value of s_k is at most $(u-1)n$, each value can be represented in $\mathrm{O}(\lg u + \lg n)$ bits. Therefore it takes $\mathrm{O}(\frac{2l}{\lg l}(\lg u + \lg n)) = \mathrm{O}(l)$ bits to store all the values s_k $(0 \le k < \frac{2l}{\lg l})$. In addition to this, we construct a look up table T storing prefix sums over every sequences which blocks can represent. In other words, given a bit vector b of length d and integer k, T stores every value of

$$T(b,k) = \sum_{i=0}^{\mathsf{rank}_1(k,b)} \mathsf{rank}_0(\mathsf{select}_1(i,b),b) \ (0 \le k < d).$$

This table takes $2^d d \lg d^2 = \mathrm{O}(\sqrt{l} \cdot \mathrm{polylog}(l))$ bits. Prefix sum for the first k terms is obtained by following procedure:

- $r \leftarrow \mathsf{select}_1(k,B), r' \leftarrow r - r\%d - 1$
- $b' \leftarrow B[r'+1, r'+2, \ldots r'+d]$
- return $s_{r/d-1} + (k - \mathsf{rank}_1(r',B))\mathsf{rank}_0(r',B) + T(b', k - \mathsf{rank}_1(r',B))$

For k-th prefix sum of conjugate sequence $\{\overline{a}\}$,

- $e_1 \leftarrow (k+1)\mathsf{access}(\{\overline{a}\},k)$
- $e_2 \leftarrow \mathsf{prefixsum}(\{a\}, n-1-\mathsf{access}(\{\overline{a}\},k))$
- return $e_1 + e_2$

The whole space usage is $\mathrm{O}(l) = \mathrm{O}(n+u)$ and both algorithms run in constant time. □

Lemma 8. *Let $\{a\} \in IMS(n,u)$ $(n \ge u)$ be the original sequence. It can be represented in $\mathrm{O}(2^k n^{1/2^k} u^{1-1/2^k})$ bits of space in total. The operation* prefixsum *over $\{a\}$ and its conjugate $\{\overline{a}\}$ can be done in $\mathrm{O}(4^k)$ time.*

Algorithm 3. $\mathsf{prefixsum}(\{a\}, i)$ (if $\{a\}$ is assigned to an internal node)

$d \leftarrow \sqrt{n/u},\ b \leftarrow i/d,\ r \leftarrow i\%d$
if $\mathsf{access}(\{a'\}, b) \neq \mathsf{access}(\{a'\}, b+1)$ **then**
  $e_1 \leftarrow \mathsf{prefixsum}(\{a''\}, v[\mathsf{access}(\{a'\}, b)]d + r)$
  $e_2 \leftarrow d \cdot \mathsf{prefixsum}(\{a'\}, b-1)$
  $e_3 \leftarrow \mathsf{prefixsum}(\{s\}, \mathsf{access}(\{a'\}, b))$
  return $e_1 + e_2 - e_3$
else
  $e_1 \leftarrow \mathsf{prefixsum}(\{a''\}, v[\mathsf{access}(\{a'\}, b)]d)$
  $e_2 \leftarrow d \cdot \mathsf{prefixsum}(\{a'\}, b-1)$
  $e_3 \leftarrow \mathsf{prefixsum}(\{s\}, \mathsf{access}(\{a'\}, b))$
  $e_4 \leftarrow r \cdot \mathsf{access}(\{a'\}, b)$
  return $e_1 + e_2 - e_3 + e_4$
end if

Algorithm 4. $\mathsf{prefixsum}(\{\overline{a}\}, i)$ (if $\{a\}$ is assigned to an internal node)

$j \leftarrow n - 1 - \mathsf{access}(\{\overline{a}\}, i)$
return $\mathsf{prefixsum}(\{a\}, j) + i(n - j - 1)$

Proof. We use the same data structure as Lemma 5. Let $\{a^{(0)}\}$ be the original sequence and construct 2^k sequences $\{a^{(2^k-1)}\}, \ldots, \{a^{(2^{k+1}-2)}\}$ which belong to $IMS(n^{1/2^k}u^{1-1/2^k}, u)$. We store these short sequences as Lemma 7. Therefore, the space complexity is shown to be $\mathrm{O}(2^k n^{1/2^k} u^{1-1/2^k})$.

Next, we consider the time complexity. Algorithms 3 and 4 shows how to calculate the prefix sum over $\{a\}$ and $\{\overline{a}\}$, respectively. There exist access operations in Algorithm 3, but this data structure does not support access. Therefore, we have to use $\mathsf{prefixsum}(\{a\}, i) - \mathsf{prefixsum}(\{a\}, i-1)$ instead of $\mathsf{access}(\{a\}, i)$. In Algorithm 3, $\mathsf{prefixsum}(\{a^{(2i+1)}\}, \cdot)$ and $\mathsf{prefixsum}(\{a^{(2i+2)}\}, \cdot)$ are called four times in total. Thus, time complexity for $\mathsf{prefixsum}(\{a\}, i)$ becomes $\mathrm{O}(4^k)$. For $\{\overline{a}\}$, it is also $\mathrm{O}(4^k)$ time. □

In the case of $n < u$, applying Lemma 8 to the conjugate sequence, the space complexity becomes $\mathrm{O}(2^k u^{1/2^k} n^{1-1/2^k})$ and $\mathsf{prefixsum}$ can be supported in $\mathrm{O}(4^k)$ time. By replacing 2^k with δ, we obtain Theorem 2.

5 A Data Structure for a Partition of an Integer

We are going to show Theorem 1.

We consider the following queries on a partition of an integer:

- $\mathsf{access}_{\mathsf{p}}(\{A\}, i) = A[i]$.
- $\mathsf{bound}_{\mathsf{p}}(\{A\}, i) = \#\{j \mid A_j > i\} = \overline{A}[i]$.
- $\mathsf{prefixsum}_{\mathsf{p}}(\{A\}, i) = \sum_{l=0}^{i} A[l]$.

Encoding this sequence as Lemma 3, it supports operations in constant time but takes $n + o(n)$ bits. It is quite larger than the information theoretic lower bound, since this method ignores the condition $\sum_{i=0}^{k-1} A[i] = n$.

For convenience, we define $A[i] = 0$ $(n > i \geq k)$. First, the following holds by definition.

Fact 3. *$A[i] \leq \sqrt{n}$ if $i > \sqrt{n}$.*

In order to support the three operations $\mathsf{access_p}, \mathsf{bound_p}, \mathsf{prefixsum_p}$ on the partition, we divide the original partition $\{A\}$ into two sequences $\{a^1\}$ and $\{a^2\}$:

- $\{a^1\} \in DMS(n, \sqrt{n})$; $a^1[k] = \min(\sqrt{n}, A[k])$.
- $\{a^2\} \in DMS(n, \sqrt{n})$; $a^2[k] = \min(\sqrt{n}, \overline{A}[k])$.

Fact 4. $\displaystyle\sum_{k=0}^{n-1} a^1[k] \leq n, \ \sum_{k=0}^{n-1} a^2[k] \leq n.$

In order to prove Theorem 1, we define the set of monotonic sequence $SMS(n)$.

Definition 3. $SMS(n) = \{\{a\} \in DMS(n, \sqrt{n}) \mid \sum_{l=0}^{n-1} a[l] \leq n\}$.

From Fact 4, two sequences $\{a^1\}$ and $\{a^2\}$ belong to $SMS(n)$. We are going to show that $\mathsf{access}, \mathsf{bound}, \mathsf{prefixsum}$ on the sequences which belong to $SMS(n)$ can be done in constant time by a data structure which uses $O(\sqrt{n})$ bits space. In order to distinguish from the operations on $DMS(n, u)$ and $IMS(n, u)$, we denote the operations on $SMS(n)$ as $\mathsf{access_s}, \mathsf{bound_s}, \mathsf{prefixsum_s}$.

Applying Theorem 2 to a sequence $\{a\} \in SMS(n)$, we obtain a data structure with $O(n^{1/2+\delta})$ bit space for any positive constant δ and supporting the three types of operations in constant time. We are going to reduce the space complexity to $O(\sqrt{n})$ bits.

First, divide the sequence $\{a\}$ into $\sqrt{n}$ blocks of length $\sqrt{n}$. Let the sequence $\{h\}$ be $h[i] = a[\sqrt{n}i] - a[\sqrt{n}(i+1)]$.

Lemma 9. $\#\{i \mid h[i] \geq j\} = O(n^{1/4} j^{-1/2})$

Proof. In order to maximize $l = \#\{i \mid h[i] > j\}$, we should set $h[0] = h[1] = \cdots = h[l-1] = j$, $h[l] = h[l+1] = \cdots = h[\sqrt{n}-1] = 0$. Considering the condition $\sum_{i=0}^{n-1} a[i] \leq n$, l should satisfy $\frac{l(l-1)\sqrt{n}j}{2} \leq n$. It means $l = O(n^{1/4} j^{-1/2})$. □

Corollary 2. $\#\{j \mid h[j] > 0\} = O(n^{1/4})$

We call a block B_j such that $h[j] = 0$ as an *empty block*, otherwise a *non-empty block*.

Lemma 10. *The k-th largest value of the sequence $\{h\}$ is* $O\left(\frac{\sqrt{n}}{k^2}\right)$.

Proof. In order to maximize $r =$ (the k-th largest value of the sequence $\{h\}$). It is equivalent to $O(n^{1/4} r^{-1/2}) > k$, thus r satisfies $r = O\left(\frac{\sqrt{n}}{k^2}\right)$. □

For each block which satisfies $h[i] > 0$, we construct the sequence $\{d_i\}$ where $d_i[j] = a[\sqrt{n}\, i + j] - a[\sqrt{n}(i+1)]$. Since $\{d_i\} \in DMS(\sqrt{n}, h[i])$, this sequence can be represented in $\mathrm{O}(2^k n^{1/2^k} h[i]^{1-1/2^k})$ bits of space for any positive constant k as Lemmas 5 and 8. The sum of space usage over all non-empty blocks is

$$\sum_{i=0}^{\sqrt{n}} \mathrm{O}(2^k n^{1/2^k} h[i]^{1-1/2^k}) \leq 2^k \zeta\left(2 - \frac{1}{2^{k-1}}\right) \mathrm{O}\left(\sqrt{n}\right), \tag{1}$$

where $\zeta(\cdot)$ is Riemann zeta function. The term $2^k\zeta(2-1/2^{k-1})$ becomes constant when $k \geq 2$.

In addition we have to store the sequence $\{c\}$ defined as $c[i] = a[\sqrt{n}i]$. Because $\{c\} \in DMS(\sqrt{n}, \sqrt{n})$, this sequence can be compressed into $\mathrm{O}(\sqrt{n})$ bit space and we can access an arbitrary term in constant time as Lemma 3. A bit vector BV is defined so that i-th bit is 1 if and only if i-th block B_i is not empty. Thus, $\mathsf{rank}_1(BV, k)$ means the number of non-empty blocks out of the first k blocks.

The whole space usage is $\mathrm{O}(\sqrt{n})$ bits and time complexity for $\mathsf{access}_\mathsf{s}(i)$ is $\mathrm{O}(2^k) = \mathrm{O}(1)$ since we fix k as a constant.

Algorithms 5 and 6 shows the procedure of operation $\mathsf{access}_\mathsf{s}$ and $\mathsf{bound}_\mathsf{s}$ on the sequence $\{a\}$, respectively.

Algorithm 5. $\mathsf{access}_\mathsf{s}(\{a\}, i)$

$b \leftarrow i/\sqrt{n},\ r \leftarrow i\%\sqrt{n}$
return $c[b+1] + d_b[r]$

Algorithm 6. $\mathsf{bound}_\mathsf{s}(\{a\}, i)$

$j \leftarrow \max\{k \mid \overline{c}[k] > i\},\ h \leftarrow i - \overline{c}[j+1],\ \mathit{diff} \leftarrow \max\{r \mid \overline{d_j}[r] \geq h\}$
return $(j+1)\sqrt{n} - \mathit{diff}$

For $\mathsf{prefixsum}_\mathsf{s}$, we prepare an additional sequence $\{g\}$ of length $\mathrm{O}(n^{1/4})$. Here $g[i]$ is defined as $g[k] = \sum_{i=0}^{k} s[i]$ where $s[i]$ =(the sum of all terms in i-th non-empty block).

Note that the number of non-empty blocks is $\mathrm{O}(n^{1/4})$ (Corollary 2). Storing this sequence explicitly, it only takes $\mathrm{O}(n^{1/4} \lg n) = \mathrm{o}(\sqrt{n})$ bits. We represent all non-empty blocks using the data structure of Theorem 1, therefore the whole space complexity of representing blocks is same as Ineq. (1). This means the space complexity of the data structure for $\{a\}$ is $\mathrm{O}(\sqrt{n})$ in total. We construct the same one for the sequences $\{a^1\}$ and $\{a^2\}$.

Algorithm 7. $\mathsf{prefixsum_s}(\{a\}, i)$

$b \leftarrow i/\sqrt{n},\ r \leftarrow i\%\sqrt{n},\ e_1 \leftarrow \sqrt{n} \cdot \mathsf{prefixsum}(\{c\}, b-1),\ e_2 \leftarrow g[\mathsf{rank}_1(BV, b)]$
$e_3 \leftarrow c[b] \cdot r$
if b-th block is non-empty **then**
  return $e_1 + e_2 + e_3 + \mathsf{prefixsum}(\{d_b\}, r)$
else
  return $e_1 + e_2 + e_3$
end if

Algorithm 7 shows how to calculate $\mathsf{prefixsum_s}(i)$.

Using the algorithms above, three operations $\mathsf{access_p}, \mathsf{bound_p}, \mathsf{prefixsum_p}$ on a partition can be represented as:

$$\mathsf{access_p}(\{A\}, i) = \begin{cases} \mathsf{bound_s}(\{a^2\}, i) & (i < \sqrt{n}), \\ \mathsf{access_s}(\{a^1\}, i) & (i \geq \sqrt{n}), \end{cases}$$

$$\mathsf{bound_p}(\{A\}, i) = \begin{cases} \mathsf{access_s}(\{a^2\}, i) & (i < \sqrt{n}), \\ \mathsf{bound_s}(\{a^1\}, i) & (i \geq \sqrt{n}), \end{cases}$$

$$\mathsf{prefixsum_p}(\{A\}, i) = \begin{cases} \mathsf{prefixsum_s}(\{a^2\}, j) + i(n-j-1) & (i < \sqrt{n}), \\ \mathsf{prefixsum_s}(\{a^1\}, i) + \mathit{diff} & (i \geq \sqrt{n}), \end{cases}$$

where

$$j = \max\{k \mid A[k] \leq i\} = n - 1 - \mathsf{bound_p}(\{A\}, i),$$

$$\mathit{diff} = \sum_{k=0}^{\sqrt{n}-1} \max(0, A[k] - \sqrt{n}).$$

The value j can be found in constant time, and *diff* can be stored explicitly. Thus, three operations are supported in constant time with $\mathrm{O}(\sqrt{n})$ bits of space. Consequently, Theorem 1 is obtained.

6 Conclusion

In this paper we have shown how to support operations $\mathsf{access_p}, \mathsf{bound_p}, \mathsf{prefixsum_p}$ on a partition of an integer. In particular, we constructed a data structure supporting constant time operations for these types of queries and use only $\mathrm{O}(\sqrt{n})$ bits of space. In order to achieve this space and time complexity, we had to construct an efficient data structure for monotonic sequences which can be used for any relation of n and u (n denotes the number of terms and u is the size of the universe). Consequently, we devised a new data structure for monotonic sequences which uses $\mathrm{O}(\min\{\frac{1}{\delta}u\left(\frac{n}{u}\right)^{\delta}, \frac{1}{\delta}n\left(\frac{u}{n}\right)^{\delta}\})$ bits and operates in $\mathrm{O}(\delta)$ or $\mathrm{O}(\delta^2)$ time. Though they are not optimal in space as long as δ is set constant, they work for arbitrary n, u.

Our data structure cannot support the operation $\mathsf{prefixsum\_bound_p}(\{A\}, i)$ on a partition, which returns $\min\{k \mid \mathsf{prefixsum_p}(\{A\}, k) > i\}$.

As future work, we consider the following:

- support the operation $\mathsf{prefixsum\_bound_p}$.
- construct a succinct data structure for a partition of integer n, that is, a data structure whose size matches the lower bound in Theorem 4.

References

1. Elias, P.: Efficient storage and retrieval by content and address of static files. J. ACM **21**(2), 246–260 (1974)
2. El-Zein, H., Lewenstein, M., Munro, J.I., Raman, V., Chan, T.M.: On the succinct representation of equivalence classes. Algorithmica **78**(3), 1020–1040 (2017)
3. Fano, R.M.: On the number of bits required to implement an associative memory. Project MAC, Massachusetts Institute of Technology (1971)
4. Fulton, W.: Young Tableaux. Cambridge University Press, Cambridge (2012)
5. Golynski, A., Orlandi, A., Raman, R., Rao, S.S.: Optimal indexes for sparse bit vectors. Algorithmica **69**(4), 906–924 (2014)
6. Hardy, G.H., Ramanujan, S.: Asymptotic formulae in combinatory analysis. Proc. Lond. Math. Soc. **2**(1), 75–115 (1918)
7. Munro, J.I., Raman, R., Raman, V., Rao, S.S.: Succinct representation of permutation and functions. Theor. Comput. Sci. **438**(22), 74–88 (2012)
8. Pibiri, G.E., Venturini, R.: Dynamic Elias-Fano representation. In: 28th Annual Symposium on Combinatorial Pattern Matching (CPM 2017), vol. 78, issue 30, pp. 1–14 (2017)
9. Raman, R., Raman, V., Rao, S.S.: Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. ACM Trans. Algorithms **3**(4) (2007). Article No. 43

How Far From a Worst Solution a Random Solution of a k CSP Instance Can Be?

Jean-François Culus[1] and Sophie Toulouse[2(✉)]

[1] CEREGMIA, Université des Antilles, Pointe-à-Pitre, France
jean-francois-culus@espe-martinique.fr
[2] LIPN (UMR CNRS 7030), Institut Galilée,
Université Paris 13, Villetaneuse, France
sophie.toulouse@lipn.univ-paris13.fr

Abstract. Given an instance I of an optimization constraint satisfaction problem (CSP), finding solutions with value at least the expected value of a random solution is easy. We wonder how good such solutions can be. Namely, we initiate the study of ratio $\rho_E(I) = (\mathrm{E}_X[v(I,X)] - \mathrm{wor}(I))/(\mathrm{opt}(I) - \mathrm{wor}(I))$ where $\mathrm{opt}(I)$, $\mathrm{wor}(I)$ and $\mathrm{E}_X[v(I,X)]$ refer to respectively the optimal, the worst, and the average solution values on I. We here focus on the case when the variables have a domain of size $q \geq 2$ and the constraint arity is at most $k \geq 2$, where k, q are two constant integers. Connecting this ratio to the highest frequency in orthogonal arrays with specified parameters, we prove that it is $\Omega(1/n^{k/2})$ if $q = 2$, $\Omega(1/n^{k-1-\lfloor \log_{p^\kappa}(k-1) \rfloor})$ where p^κ is the smallest prime power such that $p^\kappa \geq q$ otherwise, and $\Omega(1/q^k)$ in $(\max\{q,k\}+1\})$-partite instances.

Keywords: Average differential ratio
Optimization constraint satisfaction problems · Orthogonal arrays

1 Introduction

Given an integer $q \geq 2$, an optimization *Constraint Satisfaction Problem* (CSP) over $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ considers a set $\{x_1, \ldots, x_n\}$ of $\mathbb{Z}_q$-valued variables and a set $\{C_1, \ldots, C_m\}$ of constraints, where a constraint consists of the application of a (non constant) predicate $P_i : \mathbb{Z}_q^{k_i} \to \{0,1\}$ to a tuple $x_{J_i} = (x_{i_1}, \ldots, x_{i_{k_i}})$ of variables. The goal is then to assign values to the variables so as to satisfy either as many, or as few constraints as possible. For instance, in the *Maximum Satisfiability Problem* (Max Sat), the goal is to satisfy as many disjunctive clauses as possible. In Min Lin−q, the goal is to satisfy as few equations of a system of linear equations modulo q as possible.

Most often, a positive weight w_i is associated with each constraint C_i. Given a positive integer N, we represent by $[N]$ the discrete interval $\{1, \ldots, N\}$. The goal is then to optimize an objective function of the form

$$\sum_{i=1}^m w_i C_i = \sum_{i=1}^m w_i P_i(x_{J_i}) = \sum_{i=1}^m w_i P_i(x_{i_1}, \ldots, x_{i_{k_i}})$$

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 374–386, 2018.
https://doi.org/10.1007/978-3-319-94667-2_31

over $\mathbb{Z}_q^n$ where for all $i \in [m]$, $k_i \in [n]$, $P_i : \mathbb{Z}_q^{k_i} \to \{0,1\}$, $J_i = (i_1, \ldots, i_{k_i}) \subseteq [n]$ and $w_i > 0$. The special case when functions P_i all belong to a specific family $\mathcal{F}$ of functions is referred to as $\mathsf{CSP}(\mathcal{F})$ in the literature. Given a predicate P, the restriction to constraints of the form $P(x_{J_i} + v_i)$ where v_i is a constant vector is referred to as $\mathsf{CSP}-\mathsf{P}$. For example, given a positive integer k, XOR^k is the k-ary boolean predicate that accepts entries $(y_1, \ldots, y_k)$ with $y_1 + \ldots + y_k \equiv 1 \bmod 2$; then $\mathsf{Lin}-2 = \mathsf{CSP}-\{\mathsf{XOR^k} \mid \mathsf{k} \in \mathbb{N}^*\} = \mathsf{CSP}(\{\mathsf{XOR^k}, 1 - \mathsf{XOR^k} \mid \mathsf{k} \in \mathbb{N}^*\})$.

We here consider the slight generalization where functions P_i may be real-valued. In the sequel, we denote by $\mathsf{CSP}-\mathsf{q}$ the corresponding problem, and by $\mathsf{k\,CSP}-\mathsf{q}$ (*resp.*, $\mathsf{Ek\,CSP}-\mathsf{q}$) the special case when every constraint depends on at most (*resp.*, exactly) k variables, where k is a universal constant integer. As $\mathsf{k\,CSP}-\mathsf{q}$ is $\mathbf{NP}-\mathbf{hard}$ even in case when $q = k = 2$ [1], a major issue as regards optimization CSPs consists in charactering their approximation degree.

1.1 Around the Average Solution Value

Thereafter, given an instance I of $\mathsf{CSP}-\mathsf{q}$, we denote by $v(I,.)$ its objective function, by $w(I) = \sum_{i=1}^m w_i$ the total weight on I, by $\mathrm{opt}(I)$ and $\mathrm{wor}(I)$ respectively the optimum and the worst solution values on I. It is convenient to think of the average solution value on I as the expected value $\mathrm{E}_X[v(I,X)]$ of a random solution where $X = (X_1, \ldots, X_n)$ is a vector of pairwise independent random variables, each uniformly distributed over $\mathbb{Z}_q$. This value expresses as

$$\mathrm{E}_X[v(I,X)] = \sum_{i=1}^m w_i \mathrm{E}_X[P_i(X_{J_i})] = \sum_{i=1}^m w_i r_{P_i}$$

where given $i \in [m]$, r_{P_i} refers to the average value of P_i over $\mathbb{Z}_q^{k_i}$. For example, on an instance I of $\mathsf{Lin}-2$, the average solution value equals $w(I)/2$.

Solutions with value at least $\mathrm{E}_X[v(I,X)]$ are computationally easy to find, *e.g.* using the conditional expectation technique [2]. Therefore, two questions can naturally be asked: is it possible to compute within polynomial time solutions that beat the average solution value, and what is the gain of the average solution value over the worst solution value?

The former question notably leads to the concept of approximation of the *advantage over a random assignment* [3], herein referred to as *gain approximation*. The advantage of a given solution x over a random assigment is the difference $v(I,x) - \mathrm{E}_X[v(I,X)]$ if one maximizes, $\mathrm{E}_X[v(I,X)] - v(I,x)$ if one minimizes. Given $\rho \in]0,1]$, x approximates the optimum gain $|\mathrm{opt}(I) - \mathrm{E}_X[v(I,X)]|$ over $\mathrm{E}_X[v(I,X)]$ within factor ρ *iff* x achieves a *gain ratio*

$$\rho_G(I,x) = \frac{v(I,x) - \mathrm{E}_X[v(I,X)]}{\mathrm{opt}(I) - \mathrm{E}_X[v(I,X)]} \tag{1}$$

at least ρ. A given CSP Π is *ρ-gain approximable* (where ρ possibly depends on parameters of the considered instance I) if it is possible to compute within polynomial time on every instance I of Π a solution with gain ratio at least $\rho(I)$. The corresponding issue consists in determining "the best" ρ for which Π is ρ-gain approximable. Håstad and Venkatesh introduced in [3] this approximation

measure motivated by the fact that for numerous predicates P, for all constant $\varepsilon > 0$, finding solutions with value at least $(r_P + \varepsilon)w(I)$ on almost satisfiable instances of $\mathsf{Max\,CSP-P}$ is **NP−hard**. For example, for all $k \geq 3$, XOR^k is such a predicate [4]. Thus for all $k \geq 3$, $\mathsf{k\,Lin-2}$ is **NP−hard** to approximate to within any constant gain factor. It is, though, approximable within an expected gain factor of $\Omega(\sqrt{1/m})$ [3].

The latter question leads to the notion of *advantage of a random solution over the worst solution value*. Namely, we say that the average solution value on I is ρ-approximate if the ratio

$$\rho_E(I) = \frac{\mathrm{E}_X[v(I,X)] - \mathrm{wor}(I)}{\mathrm{opt}(I) - \mathrm{wor}(I)} \tag{2}$$

of the gain of $\mathrm{E}_X[v(I,X)]$ over $\mathrm{wor}(I)$ to the diameter of I is at least ρ. Given a CSP Π, we say that the average solution value is ρ-approximate for Π provided that $\rho_E(I) \geq \rho(I)$ holds for all instances I of Π. The issue here consists in exhibiting "the tightest" possible lower bound ρ for ρ_E. For example, on an instance I of $\mathsf{E3\,Lin-2}$, as given any x, any equation is satisfied either by x or by its complement, we have $\mathrm{opt}(I) + \mathrm{wor}(I) = w(I) = 2 \times \mathrm{E}_X[v(I,X)]$ [5]. Equivalently, $\rho_E(I)$ is equal to $1/2$. By contrast, for $\mathsf{E2\,Lin-2}$, we show that $\rho_E(I) \in \Omega(1/n)$, while there are satisfiable instances I on which $\rho_E(I) \in \Theta(1/n)$.

Figure 1 pictures the quantities involved in ratios (1) and (2). The two questions are complementary, and the latter has potential to enlighten the former. For instance, we may think that the more $\mathrm{E}_X[v(I,X)]$ does a good job at getting away from $\mathrm{wor}(I)$, the more it is computationally difficult to get away from it.

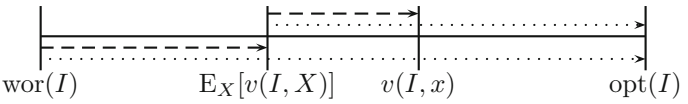


Fig. 1. Quantities involved in $\rho_G(I,x)$ and $\rho_E(I)$.

1.2 Outline

Our goal is to estimate $\rho_E(I)$ on instances I of $\mathsf{k\,CSP-q}$ given two integers $q \geq 2, k \geq 2$. To the best of our knowledge, such a study has not been carried out so far. We may nevertheless make mention of a result due to Feige et al., and that concerns the restriction to *submodular functions*. Given a positive integer n, a function $P : \{0,1\}^n \to \mathbb{R}$ is submodular *iff* it satisfies:

$$P(y) + P(z) \geq P(y_1 \vee z_1, \ldots, y_n \vee z_n) + P(y_1 \wedge z_1, \ldots, y_n \wedge z_n),\ y, z \in \{0,1\}^n$$

As shown in [6], given any maximizer x^* of such a function P, we have:

$$\mathrm{E}_X[P(X)] \geq P(x^*)/4 + P(\bar{x}_1^*, \ldots, \bar{x}_n^*)/4 + P(0,\ldots,0)/4 + P(1,\ldots,1)/4 \tag{3}$$

Since a conical combination of submodular pseudo-boolean functions is submodular, it follows from (3) that, on an instance I of $\mathsf{Max\,CSP-2}$ in which functions P_i all are submodular, we have $\mathrm{E}_X[v(I,X)] \geq \mathrm{opt}(I)/4 + 3 \times \mathrm{wor}(I)/4$ and thus, $\rho_E(I) \geq 1/4$.

In $\mathsf{k\,CSP-q}$, thought, the functions that occur in the constraints are only required to have arity at most k. In order to exhibit lower bounds for $\rho_E(I)$, we seek expressions of $\mathrm{E}_X[v(I,X)]$ as a conical combination of the optimum solution value and as few other solution values as possible. We adopt kind of a neighbourhood approach: we associate with each solution x of I a multisubset $S(I,x)$ of solutions with the same average solution value as the whole solution set, of relatively small size, and in which x occurs at least once. In the next section, we show how to derive such solution multisets from hypothetical orthogonal arrays (see Definition 2) with specified parameters that rely on characteristics of I (main theorem). In Sect. 3, we derive lower bounds for ρ_E either from orthogonal arrays of the literature, or by reduction to CSPs over a bigger alphabet. These bounds are summarized in Table 1. In the last section, we briefly discuss the obtained results and perspectives for further research.

Table 1. Lower bounds for ρ_E on instances of $\mathsf{k\,CSP-q}$ given two integers $q \geq 2, k \geq 2$: p^κ refers to the smallest prime power greater than or equal to q.

q	k	restriction	ρ_E
≥ 2	≥ 2	$(k+1)$-partite instances of $\mathsf{k\,CSP-q}$	$1/q^k$
≥ 2	$< p^\kappa$	$(p^\kappa+1)$-partite instances of $\mathsf{k\,CSP-q}$	$\Omega(1/q^k)$
≥ 3	$= 3$	$(2^{\lceil \log_2 q \rceil}+2)$-partite instances of $\mathsf{3\,CSP-q}$	$\Omega(1/q^3)$
$= 2$	≥ 2	$\mathsf{k\,CSP-2}$	$\Omega(1/n^{\lfloor k/2 \rfloor})$
≥ 3	≥ 2	$\mathsf{k\,CSP-q}$	$\Omega(1/n^{k-1-\lfloor \log_{p^\kappa}(k-1) \rfloor})$

2 Seeking Symmetries in the Solution Set

Let us start with a simple example. Given an integer $q \geq 2$, we denote by $\mathcal{O}_q$ the set of functions $P : \mathbb{Z}_q^k \to \mathbb{R}$ with $k \in \mathbb{N}^*$ that satisfy:

$$\sum_{a=0}^{q-1} P(y_1 + a, \ldots, y_k + a)/q = r_P, \qquad y \in \mathbb{Z}_q^k \tag{4}$$

$\mathcal{O}_q$ is a natural generalization to q-ary alphabets of such boolean functions P as XOR^3 that, when using a $\{-1,1\}$-encoding of truth values, are odd. For example, the predicate on $\mathbb{Z}_q^3$ that accepts solutions to equation $y_1 + y_2 - y_3 \equiv 0 \bmod q$ is a function of $\mathcal{O}_q$. By (4), given any instance I of $\mathsf{CSP}(\mathcal{O}_\mathsf{q})$, we have:

$$v(I,x)/q + \sum_{a=1}^{q-1} v(I,(x_1 + a, , \ldots, x_n + a))/q = \mathrm{E}_X[v(I,X)], \quad x \in \mathbb{Z}_q^n \tag{5}$$

Taking (5) at a solution x with optimal value, one trivially gets that the average value is $1/q$-approximate on I. Hence, for this particular CSP, one shall define $S(I,x)$ by $S(I,x) = \{x + (a,\ldots,a) \,|\, a \in \mathbb{Z}_q\}$.

2.1 Partition-Based Solution Multisets

We base our solution multisets $S(I,x)$, $x \in \mathbb{Z}_q^n$ on a specific partition $\mathcal{V} = \{V_1, \ldots, V_\nu\}$ of $[n]$. Our solution multisets then rely on an array M with ν columns and coefficients in $\mathbb{Z}_q$: given $x \in \mathbb{Z}_q^n$, each row $M_r = (M_r^1, \ldots, M_r^\nu)$ of M gives rise in $S(I,x)$ to the vector obtained from x by translating for each $c \in [\nu]$ by M_r^c all its coordinates with index in V_c. Formally, let R refer to the number of rows in the array, and let $\pi_\mathcal{V} : \mathbb{Z}_q^\nu \to \mathbb{Z}_q^n$ be defined by:

$$\pi_\mathcal{V}(u)_{V_c} = (u_c, u_c, \ldots, u_c),\ c \in [\nu]\,, \qquad u \in \mathbb{Z}_q^\nu \tag{6}$$

Then we define $S(I,.)$ by:

$$S(I,x) = (x + \pi_\mathcal{V}(M_r) \,|\, r \in [R])\,, \qquad x \in \mathbb{Z}_q^n \tag{7}$$

We more specifically are interested in pairs $(\mathcal{V}, M)$ that satisfy:

$$\textstyle\sum_{r=1}^R P_i(x_{J_i} + \pi_\mathcal{V}(M_r)_{J_i})/R = r_{P_i}, \qquad i \in [m],\ x \in \mathbb{Z}_q^n \tag{8}$$

For example, when I is an instance of $\mathsf{CSP}(\mathcal{O}_\mathsf{q})$, we may consider the pair $(\mathcal{V}, M)$ where $\mathcal{V} = \{[n]\}$ and $M = (0, 1, \ldots, q-1)^T$. Requirement (8) ensures that for all solutions x of I, the average solution value over $S(I,x)$ equals the average solution value on I. Since this holds for all $x \in \mathbb{Z}_q^n$, this in particular holds in case when x is optimal. If R^* refers to the number of times $(0, \ldots, 0)$ occurs as a row in M, then R^* precisely is the number of times x occurs in $S(I,x)$. Hence, provided that x is optimal, we have:

$$\begin{aligned} \mathrm{E}_X[v(I,X)] &= \textstyle\sum_{r=1}^R v(I, x^* + \pi_\mathcal{V}(M_r))/R \\ &= R^* \times v(I,x^*)/R + \textstyle\sum_{r \in [R] : M_r \neq (0,\ldots,0)} v(I, x^* + \pi_\mathcal{V}(M_r))/R \end{aligned}$$

We deduce that the average solution value on I is R^*/R-approximate.

Therefore, we shall seek such pairs $(\mathcal{V}, M)$ on which the ratio R^*/R is as high as possible. Observe that we may always assume that $(0, \ldots, 0)$ is a vector of highest frequency in M, due to the fact that given any $u \in \mathbb{Z}_q^\nu$, the array obtained by shifting every row of M by u still satisfies (8).

2.2 Solution Multisets Derived from Orthogonal Arrays

Given $i \in [m]$, we denote by $(c_{i,1}, \ldots, c_{i,k_i})$ the sequence of indices in $[\nu]$ such that $(i_1, \ldots, i_{k_i}) \in V_{c_{i,1}} \times \ldots \times V_{c_{i,k_i}}$. Given any $x \in \mathbb{Z}_q^n$, over $S(I,x)$, P_i is evaluated at entries

$$x_{J_i} + \pi_\mathcal{V}(M_r)_{J_i} = (x_{i_1} + M_r^{c_{i,1}}, \ldots, x_{i_{k_i}} + M_r^{c_{i,k_i}}), \qquad r \in [R]$$

$(\mathcal{V}, M)$ therefore in particular satisfies (8) at (i,x) provided that these entries coincide the same number of times with each $u \in \mathbb{Z}_q^{k_i}$. Equivalently, (8) is satisfied at (i,x) provided that each $u \in \mathbb{Z}_q^{k_i}$ occurs R/q^{k_i} times in vector multiset

$$((M_r^{c_{i,1}}, \ldots, M_r^{c_{i,k_i}}) \,|\, r \in [R]) \tag{9}$$

On the one hand, this may not occur unless indices $c_{i,1}, \ldots, c_{i,k_i}$ are pairwise distinct. On the other hand, assuming that these indices indeed are pairwise distinct, $\{c_{i,1}, \ldots, c_{i,k_i}\}$ can be any at most k-cardinality subset of $[\nu]$. These observations suggest to consider a pair $(\mathcal{V}, M)$ where $\mathcal{V}$ is a *strong coloring* of the *primary hypergraph* of I, and M is *orthogonal of strength* k:

Table 2. An $OA(27, 5, 3, 2)$ on $\mathbb{Z}_3$ (we picture the transpose).

M^1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2
M^2	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2
M^3	0	0	2	0	1	1	1	2	2	1	1	2	0	0	2	0	1	2	0	1	2	1	2	2	0	0	1
M^4	0	0	2	1	0	1	2	1	2	1	2	1	2	2	0	0	0	1	1	2	0	1	0	2	1	2	0
M^5	0	0	2	1	1	2	2	0	1	2	0	1	0	2	1	2	1	0	1	1	2	0	2	0	2	1	0

Definition 1. *The* primary hypergraph *of a CSP instance I is the hypergraph G_I where: for each $j \in [n]$, there is a vertex j in G_I; for each $i \in [m]$, there is a hyperedge $e_i = (i_1, \ldots, i_{k_i})$ in G_I. A* strong coloring *of G_I is a partition $\{U_1, \ldots, U_\nu\}$ of $[n]$ such that for all $c \in [\nu]$ and all $i \in [m]$, $|U_c \cap \{i_1, \ldots, i_{k_i}\}| \leq 1$. I is said ν-partite whenever such a partition exists.*

Definition 2. *Let $q \geq 2, t \geq 1, \nu \geq t$ and R be four integers, and Σ be a set of q symbols. Then an $R \times \nu$ array M with coefficients in Σ is an* Orthogonal Array of strength t, *$OA(R, \nu, q, t)$ in short, if given any sequence $J = (c_1, \ldots, c_t)$ of t column indices, each $v \in \Sigma^t$ occurs the same number of times as a row in subarray $M^J = (M^{c_1}, \ldots, M^{c_k})$. (See Table 2 for an illustration.)*

Since $k_i \leq k, i \in [m]$, such a pair $(\mathcal{V}, M)$ indeed satisfies for all $i \in [m]$ that the vectors of (9) coincide R/q^{k_i} times with each $u \in \mathbb{Z}_q^{k_i}$ and thus, (8).

It is possible to reduce the number of columns and possibly the strength of the array if functions P_i all satisfy, for some integer $t > 0$, that their average value when fixing any t of their variables is equal to their average value. Namely, given two integers $q \geq 2, t \geq 1$, we define $\mathcal{I}_q^t$ as the set of functions $P : \mathbb{Z}^k \to \mathbb{R}$ with $k \in \mathbb{N}^*$ that satisfy:

$$\sum_{y \in \mathbb{Z}_q^k : y_J = v} P(y)/q^{k-t} = r_P, \qquad J \subseteq [k], |J| = t, \ v \in \mathbb{Z}_q^t \tag{10}$$

Functions in $\mathcal{I}_q^t$ are known to bring inapproximability bounds for $\mathsf{k\,CSP{-}q}$. Notably, for all $k \geq 3$, if the accepting entries of a k-ary predicate $P \in \mathcal{I}_q^2$ form a subgroup of $\mathbb{Z}_q^k$, then $\mathsf{Max\,CSP{-}P}$ is **NP−hard** to approximate within any constant standard factor greater than r_P [7]. For instance, the predicate that accepts solutions to equation $y_1 + y_2 - y_3 \equiv 0 \bmod q$ over $\mathbb{Z}_q^3$ is such a predicate.

Thus assume that $P_i \in \mathcal{I}_q^t, i \in [m]$ holds for some $t > 0$. When this occurs, it is possible to fix up to t coordinates of $\pi_{\mathcal{V}}(M_r)_{J_i}$, and still obtain when averaging $P_i(x_{J_i} + \pi_{\mathcal{V}}(M_r)_{J_i})$ over $r \in [R]$ the average value of P_i. Hence, rather than a

strong coloring $\{U_1, \ldots, U_\nu\}$ of G_I, we consider partition $\mathcal{V} = \{U_1, \ldots, U_{\nu-t}, V_0\}$ where $V_0 = U_{\nu-t+1} \cup \ldots \cup U_\nu$ of $[n]$. Array M consequently has $\nu - t + 1$ columns. Given $i \in [m]$, we denote by K_i the set $\{c_{i,1}, \ldots, c_{i,k_i}\} \cap \{1, \ldots, \nu - t\}$ of column indices, by s_i it cardinality. For the sake of clarity, we assume *w.l.o.g.* that $K_i = \{c_{i,1}, \ldots, c_{i,s}\}$. Over $S(I, x)$, P_i is evaluated at entries

$$(x_{i_1} + M_r^{c_{i,1}}, \ldots, x_{i_{s_i}} + M_r^{c_{i,s_i}}, x_{i_{s_i+1}} + M_r^{\nu-t+1}, \ldots, x_{i_{k_i}} + M_r^{\nu-t+1}),\ r \in [R]$$

where $k_i - s_i \leq t$ and $s_i \leq \min\{\nu - t, k_i\} \leq \min\{\nu - t, k\}$. We deduce that setting *e.g.* $M = N \times \{0\}$ where N is an orthogonal array of strength $\min\{\nu - t, k\}$ on $\nu - t$ columns, one obtains a pair $(\mathcal{V}, M)$ that satisfies (8).

Thereafter, given an instance I of $\mathsf{CSP}-\mathsf{q}$, we denote by ν_I the *strong chromatic number* of G_I (that is, ν_I is the smallest integer such that I is ν_I-partite), and by t_I the greatest integer such that $P_i \in \mathcal{I}_q^{t_I}, i \in [m]$ (thus t_I possibly is 0). In all, the preceding discussion establishes the following connection between the average solution value on CSP instances and orthogonal arrays:

Theorem 1. *Given any three integers $q \geq 2$, $t \geq 1$ and $\nu \geq t$, we define $\rho(\nu, q, t)$ as the greatest $\rho \in [0, 1]$ for which an $OA(R, \nu, q, t)$ with highest frequency ρ exists. (For example, the $OA(27, 5, 3, 2)$ of Table 2 does achieve $\rho(5, 3, 2) = 2/27$.)*

Then for all integers $k \geq 2, q \geq 2$, the average solution value on an instance I of $\mathsf{k}\,\mathsf{CSP}-\mathsf{q}$ is $\rho(\nu_I - t_I, q, \min\{k, \nu_I - t_I\})$-approximate.

3 Approximation Quality of the Average Solution Value

3.1 From an Alphabet Size to a Greater One

We derive lower bounds for ρ_E from orthognonal arrays with a relatively small number of rows of the literature. Such arrays often require q to be a prime power. However, we can derive lower bounds for ρ_E in case when q is not a prime power by reduction to the case when q is a prime power:

Theorem 2. *Let $q \geq 3, k \geq 2$ be two integers. Then for all primes p, the average solution value on an instance I of $\mathsf{k}\,\mathsf{CSP}-\mathsf{q}$ is $\rho(\nu_I, p^{\lceil \log_p q \rceil}, k)$-approximate.*

Proof. Let $\kappa = \lceil \log_p q \rceil$, $d = p^\kappa$, and let $\pi = (\pi_1, \ldots, \pi_n)$ be a vector of surjective maps from $\mathbb{Z}_d$ to $\mathbb{Z}_q$. We interpret I as the instance $f_\pi(I)$ of $\mathsf{CSP}-\mathsf{d}$ where:

1. for each $j \in [n]$, there is in $f_\pi(I)$ a variable z_j with domain $\mathbb{Z}_d$;
2. for each $i \in [m]$, there is in $f_\pi(I)$ a constraint $P_i(\pi_{i_1}(z_{i_1}), \ldots, \pi_{i_{k_i}}(z_{i_{k_i}}))$, with the same associated weight w_i as C_i in I.

So as to retrieve solutions of I from solutions of $f_\pi(I)$, we define $g_\pi(I, .)$ by $g_\pi(I, z) = (\pi_1(z_1), \ldots, \pi_n(z_n))$, $z \in \mathbb{Z}_d^n$. By construction, $g_\pi(I, .)$ is surjective, and satisfies for all $z \in \mathbb{Z}_d^n$ that $v(I, g_\pi(I, z)) = v(f_\pi(I), z)$. The extremal solution values on I and $f_\pi(I)$ therefore satisfy:

$$\mathrm{opt}(f_\pi(I)) = \mathrm{opt}(I),\ \mathrm{wor}(f_\pi(I)) = \mathrm{wor}(I) \tag{11}$$

By contrast, $\mathrm{E}_Z[v(f_\pi(I), Z)]$ may differ from $\mathrm{E}_X[v(I, X)]$, due to the fact that two distinct vectors $x, x' \in \mathbb{Z}_q^n$ may be the image by $g_\pi(I, .)$ of a distinct number of vectors of $\mathbb{Z}_d^n$. Hence, rather than a single vector π, we consider a vector $\Pi = (\Pi_1, \ldots, \Pi_n)$ of random maps that are independently and uniformly distributed over the set $\mathcal{M}$ of surjective maps from $\mathbb{Z}_d$ to $\mathbb{Z}_q$.

Let $j \in [n]$, $a \in \mathbb{Z}_d$, $b \in \mathbb{Z}_q$, $b' \in \mathbb{Z}_q \backslash \{b\}$. Consider then function σ on $\mathcal{M}$ that maps any $\pi \in \mathcal{M}$ to the map $\sigma(\pi) : \mathbb{Z}_d \to \mathbb{Z}_q$ defined by $\sigma(\pi)(c) = b'$ if $\pi(c) = b$, b if $\pi(c) = b'$, and $\pi(c)$ otherwise. σ clearly is a bijection on $\mathcal{M}$. Hence, we have:

$$|\{\pi \in \mathcal{M} \,|\, \pi(a) = b\}| = |\{\pi \in \mathcal{M} \,|\, \sigma(\pi)(a) = b'\}| = |\{\pi \in \mathcal{M} \,|\, \pi(a) = b'\}|$$

Since $\sum_{b=0}^{q-1} \mathrm{P}_{\Pi_j}[\Pi_j(a) = b] = 1$ holds for all $j \in [n]$ and all $a \in \mathbb{Z}_d$, we first deduce that probabilities $\mathrm{P}_{\Pi_j}[\Pi_j(a) = b]$ all are equal to $1/q$. We then deduce that probabilities $\mathrm{P}_\Pi[g_\Pi(I, z) = x]$, $z \in \mathbb{Z}_d^n$, $x \in \mathbb{Z}_q^n$ all are equal to $\prod_{j=1}^n \mathrm{P}_{\Pi_j}[\Pi_j(z_j) = x_j] = 1/q^n$. Accordingly, given any $z \in \mathbb{Z}_d^n$, we have:

$$\mathrm{E}_\Pi[v(I, g_\Pi(I, z))] = \textstyle\sum_{x \in \mathbb{Z}_q^n} v(I, x) \times \mathrm{P}_\Pi[g_\Pi(I, z) = x] = \mathrm{E}_X[v(I, X)]$$

We eventually deduce that the expected average solution value on $f_\Pi(I)$ satisfies:

$$\mathrm{E}_\Pi\left[\mathrm{E}_Z[v(f_\Pi(I), Z)]\right] = \mathrm{E}_Z\left[\mathrm{E}_\Pi[v(I, g_\Pi(I, Z))]\right] = \mathrm{E}_X[v(I, X)] \tag{12}$$

By (12), there exists $\pi_* \in \mathcal{M}^n$ such that $\mathrm{E}_Z[v(f_{\pi_*}(I), Z)] \leq \mathrm{E}_X[v(I, X)]$ while by (11), for such a vector π_*, we have $\rho_E(I) \geq \rho_E(f_{\pi_*}(I))$. Since the supports of the constraints are unchanged by $f_\pi, \pi \in \mathcal{M}^n$, the result follows from Theorem 1. □

3.2 Deriving Bounds from Orthogonal Arrays of the Litterature

Let $q \geq 2$, $t \geq 1$ and $\nu \geq t$ be three integers. The smallest integer R such that an $OA(R, \nu, q, t)$ exists is referred to as $F(\nu, q, t)$ in the literature. The highest frequency of a word in an array that achieves $F(\nu, q, t)$ naturally is at least $1/F(\nu, q, t)$. Also observe that $F(\nu, q, k) \leq F(\nu', q, k)$ naturally holds for all integers $\nu' > \nu$. Hence, according to Theorems 1 and 2, given any instance I of $\mathsf{k}\,\mathsf{CSP}{-}\mathsf{q}$, we may exhibit lower bounds for $\rho_E(I)$ using inequalities:

$$\rho_E(I) \geq 1/F(\nu', q, \min\{\nu_I - t_I, k\}), \qquad \nu' \in \mathbb{N},\ \nu' \geq \nu_I - t_I \tag{13}$$

$$\rho_E(I) \geq 1/F(\nu', p^{\lceil \log_p q \rceil}, k), \qquad \nu', p \in \mathbb{N},\ \nu' \geq \nu_I,\ p \text{ prime} \tag{14}$$

First, we consider the case of instances with a bounded strong chromatic number. Given three integers $q \geq 2, k \geq 1, \nu \geq k$, there exists an $OA(q^k, \nu, q, k)$ provided that one of the following cases occurs (see *e.g.* [8]):

1. $\nu \in \{k, k+1\}$;
2. q is a prime power, $q > k$ and $\nu \in \{k+2, \ldots, q+1\}$;
3. $k = 3$, q is a power of 2, $q > 3$ and $\nu = q + 2$.

1. is trivial, considering $M \simeq \mathbb{Z}_q^k$ if $\nu = k$, $M \simeq \{y \in \mathbb{Z}_q^{k+1} \,|\, y_1 + \ldots + y_{k+1} \equiv 0 \bmod q\}$ otherwise. 2. and 3. are due to Bush. We deduce from (13) and (14) the following lower bounds for ρ_E in $O(\max\{q,k\})$-partite instances of $\mathsf{k\,CSP-q}$:

Corollary 1. *Let $q \geq 2, k \geq 2, \nu \geq k$ be three integers with $\nu \in O(\max\{q,k\})$ and I be a ν-partite instance of $\mathsf{k\,CSP-q}$. We denote by p^κ the smallest prime power greater than or equal to q. Then on I, $\rho_E(I)$ is bounded below by:*

1. $1/q^{\nu - t_I} > 1/q^k$ *if* $\nu < k + t_I$;
2. $1/q^k$ *if* $\nu \leq k + t_I + 1$;
3. $1/q^k$ *if q is a prime power, $q > k$ and* $\nu \leq q + t_I + 1$;
4. $1/p^{\kappa k} \geq 1/(2(q-1))^k$ *if q is not a prime power, $p^\kappa > k$ and* $\nu \leq p^\kappa + 1$;
5. $1/q^3$ *if $k = 3$, q is a power of 2, $q > 3$ and* $\nu \leq q + t_I + 2$;
6. $1/2^{3\lceil \log_2 q \rceil} \geq 1/(2(q-1))^3$ *if $k = 3$, q is not a power of 2 and* $\nu \leq 2^{\lceil \log_2 q \rceil} + 2$.

For example, on a k-partite instance I of $\mathsf{Lin-q}$ in which equations all are of the form $(x_{i_1} + \ldots + x_{i_k} \equiv \alpha_{i,0} \bmod q)$, we have $\rho_E(I) \geq 1/q$. For greater integers ν, we refer to families of orthogonal arrays that originate from infinite families of linear codes. Over the boolean alphabet, we consider dual codes of BCH codes. Namely, binary BCH codes allow for all integers $\kappa \geq 3, k \geq 1$ such that $2^\kappa - 1 \geq 2k + 1$ to construct an $OA(R, 2^\kappa - 1, 2, 2k)$ with $R \leq 2^{\kappa k}$ (see *e.g.* [8,9]). For such pairs (κ, k), we consequently have $F(2^\kappa - 1, 2, 2k) \leq 2^{\kappa k}$. Besides, as reported in [8], $F(2^\kappa, 2, 2k+1) = 2 \times F(2^\kappa - 1, 2, 2k)$. Hence, it also holds for such pairs (κ, k) that $F(2^\kappa, 2, 2k+1) \leq 2^{1+\kappa k}$. Accordingly:

Corollary 2. *Let $k \geq 2$ be an integer. Then on all instances I of $\mathsf{k\,CSP}$ with $\nu_I - t_I \geq k + 2$, $\rho_E(I)$ is bounded below by:*

1. $1/2^{\lceil \log_2(\nu_I - t_I + 1) \rceil \lfloor k/2 \rfloor} \geq 1/2^{\lfloor k/2 \rfloor} \times 1/(\nu_I - t_I)^{\lfloor k/2 \rfloor}$ *if k is even;*
2. $1/2^{1+\lceil \log_2(\nu_I - t_I) \rceil \lfloor k/2 \rfloor} \geq 1/2^{1+\lfloor k/2 \rfloor} \times 1/(\nu_I - t_I - 1)^{\lfloor k/2 \rfloor}$ *if k is odd.*

For $\mathsf{Max\,E2\,Lin-2}$, which is a special case of $2\,\mathsf{CSP}(\mathcal{I}_2^1)$, we thus obtain for ρ_E a lower bound of $1/2^{\lceil \log_2 \nu_I \rceil}$. And this is a tight bound: given $n \in \mathbb{N}^*$, let I_n refer to the instance where G_{I_n} is the complete graph K_{2n}, and equations all are of the form $(x_j + x_h \equiv 0 \bmod 2)$. I_n is trivially satisfiable by the vectors of allzeros and of allones, and $v(I_n, .)$ is minimized at balanced vectors. Hence, we have:

$$\rho_E(I_n) = (\tbinom{2n}{2}/2 - 2\tbinom{n}{2})/(\tbinom{2n}{2} - 2\tbinom{n}{2}) = 1/(2n) = 1/\nu_{I_n},\ n \in \mathbb{N}^*$$

For greater prime powers q, trace-codes of Reed-Solomon codes give rise for all integers s, k, λ such that $q^s \geq k > q^\lambda \geq 1$ to $q^{1+s(k-1-\lambda)} \times q^s$ orthogonal arrays of strength k on $\mathbb{Z}_q$ [10]. Accordingly:

Corollary 3. *Let $q \geq 3, k \geq 2$ be two integers. We denote by p^κ the smallest prime power such that $p^\kappa \geq q$, by λ the greatest integer such that $k > p^{\kappa\lambda}$. If q is a prime power, then for all instances I of $\mathsf{k\,CSP-q}$ with $\nu_I - t_I \geq k$, we have:*

$$\rho_E(I) \geq 1/q^{1+\lceil \log_q(\nu_I - t_I) \rceil (k-1-\lambda)} \geq 1/q^{k-\lambda} \times 1/(\nu_I - t_I - 1)^{k-1-\lambda}$$

Otherwise, for all instances I of $\mathsf{k\,CSP-q}$ with $\nu_I \geq k$, we have:

$$\rho_E(I) \geq 1/p^{\kappa(1+\lceil \log_{p^\kappa} \nu_I \rceil (k-1-\lambda))} \geq 1/(2(q-1))^{k-\lambda} \times 1/(\nu_I - 1)^{k-1-\lambda}$$

4 Concluding Remarks

In order to evaluate the quality of the average solution value, one shall compare the lower bounds we obtain for ρ_E to both gain and *differential approximability* bounds of the literature. The *differential approximation measure* (see [11] for an introduction) evaluates the performance of a given solution x by the ratio

$$\rho_D(I,x) = \frac{v(I,x) - \mathrm{wor}(I)}{\mathrm{opt}(I) - \mathrm{wor}(I)} \tag{15}$$

Thus $\rho_E(I)$ precisely is the *average differential ratio* on I. ρ-differential approximable problems are defined just as the same as for the gain approximation measure. Notice that by definition of $\rho_G(I,x)$, $\rho_E(I)$ and $\rho_D(I,x)$, we have:

$$\rho_D(I,x) = \rho_E(I) + (1 - \rho_E(I)) \times \rho_G(I,x) \tag{16}$$

Hence, if a given CSP Π is ρ-gain approximable, and the average solution value is δ-approximate for Π, then Π is $[\rho + \delta(1-\rho)]$-differential approximable.

We summarize in Table 3 the bounds we are aware of. In this table, we take into account the fact that gain approximability lower bounds that hold for $\mathsf{Lin-2}$ somehow extend to $\mathsf{k\,CSP-q}$ for all integers $q \geq 2, k \geq 3$:

Proposition 1. *For all integers $q \geq 2, k \geq 2$, if $(\mathsf{k}\lceil \log_2 \mathsf{q} \rceil)\mathsf{Lin-2}$ is approximable within gain factor ρ, then $\mathsf{k\,CSP-q}$ is approximable within differential factor ρ and, provided that q is a power of 2, within gain factor ρ.*

Table 3. Differential (ρ_D) and gain (ρ_G) approximability bounds for $\mathsf{k\,CSP-q}$ that are achievable by either deterministic (det.) or randomized (exp.) algorithms, and their comparison to ρ_E: p^κ refers to the smallest prime power $\geq q$; inapproximability bounds are given for all constant $\varepsilon > 0$, and assume $\mathbf{P} \neq \mathbf{NP}$.

Approximability bounds in k-partite instances of $\mathsf{Ek\,CSP-q}$

k	q	t_I	ρ_G det.	ρ_D det.	ρ_E
$=2$	$=2$	$=1$	0.561 [13]	0.78 [13]	$=1/2$
≥ 3	≥ 2	$=k-1$	$\neg\ \varepsilon$ [7]	$\neg\ 1/q + \varepsilon$ [7]	$\geq 1/q$
≥ 3	$\geq 2, \leq k$	$=2$	$\neg\ \varepsilon$ [7]	$\neg\ O(k/q^{k-1}) + \varepsilon$ [7]	$\geq 1/q^{k-2}$
≥ 3	$\geq k$	$=2$	$\neg\ \varepsilon$ [7]	$\neg\ O(k/q^{k-2}) + \varepsilon$ [7]	$\geq 1/q^{k-2}$

Gain approximability bounds for $\mathsf{Ek\,CSP-q}$

k	q	t_I	ρ_G det.	ρ_G exp.	ρ_E
$=2$	$=2$	≥ 0	$\Omega(1/\ln n)$ [14]		$\Omega(1/\nu_I)$
$=3$	$=2$	$=2$	$\Omega(1/m)$ [3]	$\Omega(\sqrt{\ln n/n})$ [15]	$=1/2$
≥ 4	$=2$	≥ 0	$\Omega(1/m)$ [3]	$\Omega(1/\sqrt{m})$ [3]	$\Omega(1/\nu_I^{\lfloor k/2 \rfloor})$
≥ 2	$=2^\kappa, \geq 4$	≥ 0	$\Omega(1/m)$	$\Omega(1/\sqrt{m})$	$\Omega(1/\nu_I^{k-1-\lfloor \log_{2^\kappa}(k-1) \rfloor})$

Other differential approximability bounds for $\mathsf{k\,CSP-q}$

k, q	ρ_D det.	ρ_D exp.	ρ_E
$k=2$ or $(k,q)=(3,2)$	$\Omega(1)$ [16,17]		$\Omega(1/\nu_I)$
$k \geq 3$ and $q \geq 3$	$\Omega(1/m)$	$\Omega(1/\sqrt{m})$	$\Omega(1/\nu_I^{k-1-\lfloor \log_{p^\kappa}(k-1) \rfloor})$

Proof (sketch). Let I be an instance of $\mathsf{k}\,\mathsf{CSP}-\mathsf{q}$. First map I to an instance J of $(\kappa\mathsf{k})\mathsf{CSP}-2$ where $\kappa = \lceil \log_2 q \rceil$ using a binary encoding of the variables. Then map J to an instance H of $(\kappa\mathsf{k})\mathsf{Lin}-2$ using the discrete Fourier transform. When $q = 2^\kappa$, the average solution values on I and J (and thus, on H) are identical. □

For such symptomatic CSPs as the restriction of $\mathsf{CSP}(\mathcal{I}_\mathsf{q}^{\mathsf{k}-1})$ to k-partite instances for all $k \geq 3$, and $\mathsf{CSP}(\mathcal{O}_\mathsf{q})$, $\mathrm{E}_X[v(I,X)]$ trivially brings the same differential approximation guarantee of $1/q$, which essentially is optimal, as in the standard approximation framework. By contrast, for $2\,\mathsf{CSP}-2$, $\mathrm{E}_X[v(I,X)]$ is of rather low quality, considering that $\Omega(1/n)$ is a tight lower bound for $\rho_E(I)$, while $2\,\mathsf{CSP}-2$ is approximable within gain and differential factor respectively $\Omega(1/\ln n)$ and $\Omega(1)$. For greater integers k, in dense instances of $\mathsf{k}\,\mathsf{CSP}-2$, the factor of $\Omega(1/n^{\lfloor k/2 \rfloor})$ $\mathrm{E}_X[v(I,X)]$ gains over $\mathrm{wor}(I)$ is significantly greater than the best gain approximation guarantee of $\Omega(1/m)$ known so far, and comparable to the best expected gain factor of $\Omega(1/\sqrt{m})$ known so far.

Starting with an $R\times\nu$-array M on $\mathbb{Z}_q$, we can define a probability distribution on $\mathbb{Z}_q^\nu$ by associating with each vector $u \in \mathbb{Z}_q^\nu$ its frequency in M. Then, M is orthogonal of strength t *iff* this distribution is *balanced t-wise independent*. A function $P : \mathbb{Z}_q^\nu \to \mathbb{R}$ with minimal value P_* similarly belongs to $\mathcal{I}_q^t$ *iff* $1/q^\nu \times (P-P_*)/(r_P-P_*)$ defines a balanced t-wise independent distribution on $\mathbb{Z}_q^\nu$. This classic notion is strongly involved in the exhibition of approximation resistant predicates (notably see [7,12]). The analysis we proposed therefore reinforces the connection between balanced t-wise independence and approximability of $\mathsf{k}\,\mathsf{CSP}-\mathsf{q}$ by allowing the establishment of positive results. Observe that the arrays we used contain no duplicated rows. Therefore, lower bounds for ρ_E in ν-partite instances could be improved by exhibiting orthogonal arrays (or balanced k-wise independent measures) that maximize their highest frequency, especially in case when $\nu \in \Theta(\max\{q,k\})$. Table 4 provides a few illustrations of this fact.

The average differential ratio has potential to provide new insights into CSPs. First, the method we used to obtain lower bounds for ρ_E not only shows that $\mathrm{E}_X[v(I,X)]$ achieves some differential ratio ρ, but also indicates that ρ-differential approximate solutions are spread all over the solution set. It thus provides additional information on the repartition of solution values. Although we took into account parameters ν_I and t_I so as to refine our analysis, apart from the arity of their constraints, we did not restrict the instances we considered. Hence, a next step should be the identification of hypergraphs and function properties that allow to build partition-based solution multisets of low cardinality that satisfy (8). More generally, it would be worthwhile to characterize such functions families $\mathcal{F}$ as the set of submodular functions for which $\mathsf{Max}\,\mathsf{CSP}(\mathcal{F})$ or $\mathsf{Min}\,\mathsf{CSP}(\mathcal{F})$ admits a constant lower bound for ρ_E. Finally, the properties of ρ_E viewed as a complexity measure, including its connections to other measures, should be investigated. Notably, because for $\mathsf{E3}\,\mathsf{Lin}-2$, $\rho_E \in O(1)$, the authors of [5] could derive from the hardness result of [4] as regards $\mathsf{E3}\,\mathsf{Lin}-2$ a constant inapproximability bound of 0 for the diameter of $3\,\mathsf{Sat}$ instances.

Table 4. Comparison of arrays that maximize their highest frequency among those that achieve $F(\nu, q, k)$ (on the right) to arrays that minimize their number of rows among those that achieve $\rho(\nu, q, k)$ (on the left). In both cases, we indicate the ratio R^*/R of the highest number of occurrences of a word to the number of rows in the array. (These arrays were calculated by computer using linear programs.)

		ν						ν					
q	k	4	5	6	7	8	9	4	5	6	7	8	9
2	2	**2/12**	**2/12**	1/8	1/8	**2/20**	**2/20**	1/8	1/8	1/8	1/8	1/12	1/12
	4	1/16	1/16	**3/80**	**4/144**	**6/240**		1/16	1/16	1/32	1/64	1/64	
	6	–	1/32	1/64	1/64	**4/448**	**6/960**	–	1/32	1/64	1/64	1/128	1/256
3	2	1/9	**2/27**	**3/45**	**3/45**			1/9	1/18	1/18	1/18		
4	2	1/16	1/16	**7/160**				1/16	1/16	1/32			

References

1. Garey, M., Johnson, D., Stockmeyer, L.: Some simplified NP-complete graph problems. Theor. Comput. Sci. **1**(3), 237–267 (1976)
2. Johnson, D.S.: Approximation algorithms for combinatorial problems. J. Comput. Syst. Sci. **9**(3), 256–278 (1974)
3. Håstad, J., Venkatesh, S.: On the advantage over a random assignment. Random Struct. Algorithms **25**(2), 117–149 (2004)
4. Håstad, J.: Some optimal inapproximability results. J. ACM **48**(4), 798–859 (2001)
5. Escoffier, B., Paschos, V.T.: Differential approximation of MIN SAT, MAX SAT and related problems. EJOR **181**(2), 620–633 (2007)
6. Feige, U., Mirrokni, V.S., Vondrák, J.: Maximizing non-monotone submodular functions. SIAM J. Comput. **40**(4), 1133–1153 (2011)
7. Chan, S.O.: Approximation resistance from pairwise-independent subgroups. J. ACM **63**(3), 27:1–27:32 (2016)
8. Hedayat, A., Sloane, N.J.A., Stufken, J.: Orthogonal Arrays: Theory and Applications. Springer, New York (1999). https://doi.org/10.1007/978-1-4612-1478-6
9. MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error-Correcting Codes. North Holland Publishing Co., Amsterdam (1977)
10. Bierbrauer, J.: Construction of orthogonal arrays. J. Stat. Plann. Infer. **56**(1), 39–47 (1996)
11. Demange, M., Paschos, V.T.: On an approximation measure founded on the links between optimization and polynomial approximation theory. Theor. Comput. Sci. **158**(1–2), 117–141 (1996)
12. Austrin, P., Håstad, J.: Randomly supported independence and resistance. SIAM J. Comput. **40**(1), 1–27 (2011)
13. Alon, N., Naor, A.: Approximating the cut-norm via grothendieck's inequality. SIAM J. Comput. **35**(4), 787–803 (2006)
14. Nemirovski, A.S., Roos, C., Terlaky, T.: On maximization of quadratic form over intersection of ellipsoids with common center. Math. Program. **86**(3), 463–473 (1999)
15. Khot, S., Naor, A.: Linear equations modulo 2 and the L_1 diameter of convex bodies. SIAM J. Comput. **38**(4), 1448–1463 (2008)
16. Nesterov, Y.: Semidefinite relaxation and nonconvex quadratic optimization. Optim. Methods Softw. **9**(1–3), 141–160 (1998)

17. Culus, J.F., Toulouse, S.: 2 CSPs all are approximable within some constant differential factor. In: 5th International Symposium on Combinatorial Optimization (ISCO) (20I8, to appear)

Author Index