

M. Sohel Rahman
Wing-Kin Sung
Ryuhei Uehara (Eds.)

LNCS 10755

WALCOM: Algorithms and Computation

12th International Conference, WALCOM 2018
Dhaka, Bangladesh, March 3–5, 2018
Proceedings

 Springer

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, Lancaster, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Friedemann Mattern

ETH Zurich, Zurich, Switzerland

John C. Mitchell

Stanford University, Stanford, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

TU Dortmund University, Dortmund, Germany

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Gerhard Weikum

Max Planck Institute for Informatics, Saarbrücken, Germany

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

M. Sohel Rahman · Wing-Kin Sung
Ryuhei Uehara (Eds.)

WALCOM: Algorithms and Computation

12th International Conference, WALCOM 2018
Dhaka, Bangladesh, March 3–5, 2018
Proceedings

Editors

M. Sohel Rahman 
Bangladesh University of Engineering
and Technology
Dhaka
Bangladesh

Ryuhei Uehara 
Japan Advanced Institute of Science
and Technology
Ishikawa
Japan

Wing-Kin Sung
National University of Singapore
Singapore
Singapore

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-75171-9 ISBN 978-3-319-75172-6 (eBook)
<https://doi.org/10.1007/978-3-319-75172-6>

Library of Congress Control Number: 2018930748

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

© Springer International Publishing AG, part of Springer Nature 2018

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

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 WALCOM 2018, the 12th International Conference and Workshop on Algorithms and Computation, held during March 3–5, 2018, at the Department of Computer Science and Engineering (CSE), Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh. The conference covered diverse areas of algorithms and computation, namely, approximation algorithms, computational geometry, combinatorial algorithms, computational biology, computational complexity, data structures, graph and network algorithms, and online algorithms. The conference was organized jointly by Bangladesh Academy of Sciences (BAS), and the Department of Computer Science and Engineering, BUET in cooperation with IEICE Technical Committee on Theoretical Foundations of Computing (COMP), the Special Interest Group for Algorithms (SIGAL) of the Information Processing Society of Japan (IPSJ) and EATCS Japan Chapter.

WALCOM is an annual conference series on all aspects of algorithms and computation and this year marked the 12th successful organization of this event, which has a special meaning and carries a strong value in our culture. WALCOM started quite humbly with a vision to patronize the less privileged researchers, especially, in the South Asian countries without compromising the scientific quality through providing an international platform to disseminate their high-quality research works. From the very inception of WALCOM, it was led by a strong Steering Committee comprising eminent and senior scientists from Bangladesh, Germany, India, Japan, Korea, and the UK and for each and every event, the technical program was finalized by selecting the highest quality papers from among those submitted through a rigorous reviewing and discussion process by the respective Program Committees comprising computer scientists of international repute from different parts of the globe. Notably, the Program Committee of WALCOM 2018 comprised 37 eminent researchers from Australia, Bangladesh, Canada, Chile, Egypt, Finland, France, Greece, Hong Kong, India, Israel, Italy, Japan, Singapore, South Africa, South Korea, Taiwan, UK and USA.

Since its inception, WALCOM has grown substantially in reputation and has been able to attract researchers and scientists around the globe. Although, initially the idea was to host WALCOM in Bangladesh and India on alternate years, recently it has gone beyond that. In particular, in 2016 and 2017, WALCOM was successfully hosted in Nepal and Taiwan, respectively, and it is planned that in 2020, it will be hosted in Singapore.

This year we could accept only 22 high-quality papers, which were selected based on thorough reviewing (at least three review reports per paper) followed by in-depth discussion sessions by the Program Committee. Following the recent tradition that was initiated in WALCOM 2015, two Best Paper Awards were also given. We are pleased to announce that “Online Facility Assignment” authored by Abu Reyan Ahmed, Md. Saidur Rahman, and Stephen Kobourov and “Boosting over Non-deterministic

ZDDs” authored by Takahiro Fujita, Kohei Hatano, and Eiji Takimoto were selected for the Best Paper Awards by the Program Committee. We are also delighted to highlight that following the tradition of the previous years, two special issues—one in the *Journal of Graph Algorithms and Applications* and the other in *Theoretical Computer Science*—are planned featuring the extended versions of selected papers from WALCOM 2018.

In addition to the 22 contributed talks, the scientific program of the workshop included invited talks by three eminent researchers, namely, Prof. Giuseppe Di Battista of Università Roma Tre, Italy, Prof. Naoki Katoh of Kwansei Gakuin University, Japan, and Prof. Limsoon Wong, National University of Singapore. 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 WALCOM 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 must 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 WALCOM 2018 in the LNCS series. As always, we are deeply indebted to the WALCOM Steering Committee for their continuous guidance, support, and leadership. Above all, we are extremely grateful to the Organizing Committee of WALCOM 2018 for making the event a grand success. Last but not the least, we express our heartiest gratitude to the sponsors, namely, IPDC Finance Ltd. and Dynamic Solutions Innovators (DSi) Ltd. for their kind and generous support.

March 2018

M. Sohel Rahman
Wing-Kin Sung
Ryuhei Uehara

Organization



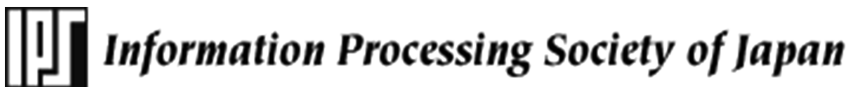
WALCOM Steering Committee

Kyung-Yong Chwa	KAIST, Korea
Costas S. Iliopoulos	KCL, UK
M. Kaykobad	BUET, Bangladesh
Petra Mutzel	TU Dortmund, Germany
Shin-ichi Nakano	Gunma University, Japan
Subhas Chandra Nandy	ISI, India
Takao Nishizeki	Tohoku University, Japan
C. Pandu Rangan	IIT, Madras, India
Md. Saidur Rahman	BUET, Bangladesh

WALCOM 2018 Organizers



WALCOM 2018 Supporters



WALCOM 2018 Program Committee

Hee-Kap Ahn	POSTECH, South Korea
Muhammad Jawaherul Alam	Amazon, USA
Md. Shamsuzzoha Bayzid	BUET, Bangladesh
Guillaume Blin	Université de Bordeaux, France
Jackie Daykin	King's College, London, UK
Amr Elmasry	Alexandria University, Egypt
Wing-Kai Hon	Nation Tsing Hua University, Taiwan
Seok-Hee Hong	University of Sydney, Australia
Giuseppe F. Italiano	University of Rome Tor Vergata, Italy
Jesper Jansson	The Hong Kong Polytechnic University, Hong Kong, SAR China
Ralf Klasing	CNRS and University of Bordeaux, France
Gad M. Landau	University of Haifa, Israel
Hon Wai Leong	National University of Singapore, Singapore
Giuseppe Liotta	University of Perugia, Italy
Stefano Lonardi	University of California, Riverside, USA
Debajyoti Mondal	University of Saskatchewan, Canada
Krishnendu Mukhopadhyaya	Indian Statistical Institute, India
Subhas Chandra Nandy	Indian Statistical Institute, India
Gonzalo Navarro	University of Chile, Chile
Solon P. Pissis	King's College, London, UK

Simon J. Puglisi	University of Helsinki, Finland
Tomasz Radzik	King's College, London, UK
Atif Hasan Rahman	BUET, Bangladesh
M. Sohel Rahman	BUET, Bangladesh
Md. Saidur Rahman	BUET, Bangladesh
C. Pandu Rangan	Indian Institute of Technology, Madras, India
Kunihiko Sadakane	The University of Tokyo, Japan
William F. Smyth	McMaster University, Canada
Paul Spirakis	University of Liverpool, UK
Wing-Kin Sung	National University of Singapore, Singapore
Etsuji Tomita	The University of Electro-Communications, Japan
Ryuhei Uehara	Japan Advanced Institute of Science and Technology, Japan
Osamu Watanabe	Tokyo Institute of Technology, Japan
Bruce Watson	Stellenbosch University, South Africa
Sue Whitesides	University of Victoria, Canada
Prudence Wong	University of Liverpool, UK
Hsu-Chun Yen	National Taiwan University, Taiwan

WALCOM 2018 Advisory Committee

Mesbahuddin Ahmed (Secretary)	BAS, Bangladesh
M. Shamsher Ali (Fellow)	BAS, Bangladesh
Naiyyum Choudhury (Fellow)	BAS, Bangladesh
Quazi Abdul Fattah (President)	BAS, Bangladesh
Saiful Islam (Vice-Chancellor)	BUET, Bangladesh
Md. Saidur Rahman	CSE, BUET, Bangladesh
M. Kaykobad	CSE, BUET, Bangladesh

WALCOM 2018 Organizing Committee

Muhammad Abdullah Adnan	CSE, BUET, Bangladesh
Ishtiyaque Ahmad	CSE, BUET, Bangladesh
Md. Benzir Ahmed	CSE, BUET, Bangladesh
Toufique Ahmed	CSE, BUET, Bangladesh
Shareef Ahmed	CSE, BUET, Bangladesh
Md. Mostofa Akbar	CSE, BUET, Bangladesh
Muhammad Rashed Alam	CSE, BUET, Bangladesh
Zahangir Alam	CSE, BUET, Bangladesh
Mohammed Eunus Ali	CSE, BUET, Bangladesh
Muhammad Masroor Ali	CSE, BUET, Bangladesh

M. Shamsher Ali	BAS, Bangladesh
Mohammad Al-Mahmud	CSE, BUET, Bangladesh
Abdus Salam Azad	CSE, BUET, Bangladesh
Md. Aashikur Rahman Azim	CSE, BUET, Bangladesh
Madhusudan Basak	CSE, BUET, Bangladesh
Md. Muradul Bashir	CSE, BUET, Bangladesh
Md. Shamsuzzoha Bayzid	CSE, BUET, Bangladesh
Md. Shariful Islam Bhuyan	CSE, BUET, Bangladesh
Naiyyum Choudhury	.
Siddhartha Shankar Das	CSE, BUET, Bangladesh
Sujoy Das	CSE, BUET, Bangladesh
Quazi Abdul Fattah	CSE, BUET, Bangladesh
Ch. Md. Rakin Haider	CSE, BUET, Bangladesh
Md. Manzurul Hasan	CSE, BUET, Bangladesh
Tanzima Hashem	CSE, BUET, Bangladesh
Mahmudur Rahman Hera	CSE, BUET, Bangladesh
Abu Sayed Md. Latiful Hoque	CSE, BUET, Bangladesh
Md. Saddam Hossain	CSE, BUET, Bangladesh
Mohammad Sajjad Hossain	CSE, BUET, Bangladesh
Md. Shohrab Hossain	CSE, BUET, Bangladesh
Anindya Iqbal	CSE, BUET, Bangladesh
Md. Monirul Islam	CSE, BUET, Bangladesh
Mohammad Mahfuzul Islam	CSE, BUET, Bangladesh
A. B. M. Alim Al Islam	CSE, BUET, Bangladesh
Md. Saiful Islam	CSE, BUET, Bangladesh
Md. Monirul Islam	CSE, BUET, Bangladesh
Md. Rezaul Karim	CSE, BUET, Bangladesh
M. Kaykobad	CSE, BUET, Bangladesh
Shahidul Islam Khan	CSE, BUET, Bangladesh
Mehnaz Tabassum Mahin	CSE, BUET, Bangladesh
M. A. Mazed	.
Md. Abul Kashem Mia	CSE, BUET, Bangladesh
Md Amir Hossain Mollah	CSE, BUET, Bangladesh
Mahjabin Nahar	CSE, BUET, Bangladesh
Muhammad Ali Nayeem	CSE, BUET, Bangladesh
Mahmuda Naznin	CSE, BUET, Bangladesh
Novia Nurain	CSE, BUET, Bangladesh
Md. Tarikul Islam Papon	CSE, BUET, Bangladesh
Adnan Quaium	CSE, BUET, Bangladesh
Md. Ishat - E - Rabban	CSE, BUET, Bangladesh
A. K. M. Ashikur Rahman	CSE, BUET, Bangladesh
Atif Hasan Rahman	CSE, BUET, Bangladesh
Md. Mizanur Rahman	CSE, BUET, Bangladesh
Md. Saidur Rahman	CSE, BUET, Bangladesh

Mohammad Saifur Rahman	CSE, BUET, Bangladesh
M. Sohel Rahman	CSE, BUET, Bangladesh
Md. Iftekharul Islam Sakib	CSE, BUET, Bangladesh
Rakib Ahmed Saleh	CSE, BUET, Bangladesh
Nazmus Saquib	CSE, BUET, Bangladesh
Md. Abdus Sattar	CSE, BUET, Bangladesh
Khaled Mahmud Shahriar	CSE, BUET, Bangladesh
Rifat Shahriyar (Secretary)	CSE, BUET, Bangladesh
Sadia Sharmin (Joint Secretary)	CSE, BUET, Bangladesh
Abida Sanjana Shemonti	CSE, BUET, Bangladesh
Shaheena Sultana	CSE, BUET, Bangladesh
Abu Wasif	CSE, BUET, Bangladesh

WALCOM 2018 Additional Reviewers

Ahn, Taehoon	Epstein, Leah
Akrida, Eleni C.	Evans, Will
Aravind, N. R.	Fukunaga, Takuro
Bae, Sang Won	Ghosh, Sasthi
Baisya, Dipankar Ranjan	Giot, Romain
Balaji, Nikhil	Giraud, Mathieu
Bampas, Evangelos	Grabowski, Szymon
Bandyapadhyay, Sayan	Gronemann, Martin
Banik, Aritra	Han, Xin
Barua, Rana	Idrees, Samah
Bentert, Matthias	Johnson, Timothy
Bhagat, Subhash	Kalyanasundaram, Subrahmanyam
Bhattacharya, Binay	Kardoš, František
Bilò, Davide	Karmakar, Arindam
Canzar, Stefan	Kawamura, Akitoshi
Charalamopoulos, Panagiotis	Khramtsova, Elena
Choi, Jongmin	Kim, Mincheol
Choi, Yujin	Kindermann, Philipp
Conte, Alessio	Kirousis, Lefteris
D'Emidio, Mattia	Kolay, Sudeshna
Das, Gautam K.	Krohn, Erik
Das, Syamantak	Laekhanukit, Bundit
De Luca, Felice	Lamprou, Ioannis
Deligkas, Argyrios	Lauri, Juho
Di Stefano, Gabriele	Lee, Seungjoon
Drineas, Petros	Mandal, Partha Sarathi
Dunkelman, Orr	Marino, Andrea
Elbassioni, Khaled	Mestre, Julian

XII Organization

Mhaskar, Neerja
Minato, Shin-Ichi
Mukhopadhyaya, Srabani
Mulzer, Wolfgang
Musco, Christopher
Nayeem, Muhammad Ali
Nicholson, Patrick K.
Nikolaev, Alexey
Oh, Eunjin
Pandey, Arti
Pisanti, Nadia
Rao Satti, Srinivasa
Reddy, Vinod
Roy, Sasanka
Salson, Mikaël

Schieber, Baruch
Schmidt, Jens M.
Seki, Shinnosuke
Shahrokhi, Farhad
Shatabda, Swakkhar
Silveira, Rodrigo
Subramani, K.
Tappini, Alessandra
Uddin, Md Yusuf Sarwar
Uno, Takeaki
Uricaru, Raluca
Valenzuela, Daniel
Viglietta, Giovanni
Wismath, Steve
Yoshinaka, Ryo

WALCOM 2018 Sponsors



Invited Talks

Optimal Sink Location Problems on Dynamic Flow Networks

Naoki Katoh

Kwansei Gakuin University
naoki.katoh@gmail.com

Abstract. In recent years, catastrophic disasters by massive natural disasters such as typhoon, earthquake, tsunami, volcano eruption have been increasing in the world, and disaster management is becoming extremely important more than ever. For example, in the Tohoku-Pacific Ocean Earthquake that happened in Japan on March 11, 2011, serious damage was caused by a tsunami. Although disaster prevention in civil and architectural engineering fields in Japan has been considered previously mainly from physical aspects, it is difficult to prevent large tsunamis physically. Therefore, disaster prevention from non-physical aspects, such as city planning and evacuation planning, has become more important recently.

In particular, to reduce the deaths due to tsunami is critically important on coastal areas of Japan. For this, we need to prepare evacuation building so that people can protect their lives. Such problems are formulated by the use of a dynamic network which consists of a graph that models a road network in which a capacity as well as a transit time is associated with each edge, and asks to find a way to evacuate evacuees originally existing at vertices to facilities (evacuation centers) as quickly as possible. The problem can be viewed as a generalization of classical k -center and k -median problems. We shall show recent results about the difficulty and approximability of a single-facility location for general networks and polynomial time algorithms for k -facility location problems in path and tree networks. We also mention the minimax regret version of these problems, and multi-commodity dynamic flow problems.

Keywords: Disaster prevention • Disaster management • Evacuation
 k -center problem • k -median problems • Facility location
Multi-commodity dynamic flow problem

Logic in Computational Biology

Limsoon Wong 

National University of Singapore
13 Computing Drive, 117417, Singapore
wongls@comp.nus.edu.sg

Abstract. I will describe some problem-solving principles that are common to multiple types of problems, even in different disciplines. I will illustrate using different areas in computer science, medicine, biology, and biotechnology. These principles are simple logical ways to exploit fundamental properties of each problem domain, highlighting the value of both logical thought and domain knowledge, and bringing out the sometimes creative way of applying the former to the latter in the context of each problem being solved. In the specific context of computational biology, I will discuss the use of deductive, abductive, and inductive inference in de-noising protein-protein interaction networks, identifying homologous proteins, inferring key mutations, and diagnosing specific pediatric leukemias. In the course of this discussion, I will illustrate also the useful tactics of fixing violation of invariants and guilt by association. As a demonstration of the universality of logic as a scientific problem-solving paradigm, I will show parallels in common computer science applications (e.g. deriving a better database design and securing computers against rootkit attacks).

Keywords: Computational biology • Logical inference from invariants
Violation of invariants • Guilt by association

Morphing Planar Graph Drawings

Giuseppe Di Battista

Roma Tre University, Rome, Italy
giuseppe.dibattista@uniroma3.it

Abstract. Given two drawings Γ_0 and Γ_1 of the same graph G a *morph* between Γ_0 and Γ_1 is a continuously changing family of drawings of G indexed by time $t \in [0, 1]$, such that the drawing at time $t = 0$ is Γ_0 and the drawing at time $t = 1$ is Γ_1 .

Suppose that both Γ_0 and Γ_1 have a certain geometric property. E.g. they are planar, their edges are straight-line segments, or their edges are polygonal lines composed of horizontal and vertical segments. It is interesting, both from the theoretical and from the applications perspectives, that all the drawings of the morph *preserve* that property.

The problem of finding a morph between two drawings of the same graph that preserves one or more properties is not trivial and attracted the attention of several researchers since the first half of the twentieth century. As an example, Cairns in 1944 proved that a morph that preserves planarity of a triangulation always exists. Thomassen in 1983 extended the result to all planar straight-line drawings of embedded graphs.

Morphs that preserve a certain property can be classified from several perspectives. As an example they can be different in terms of vertex trajectories, in terms of vertex speed, in terms of number of steps, or in terms of arithmetic precision that is needed to compute the position of the geometric components of the drawings.

We survey the state-of-the-art on this intriguing topic focusing the attention on morphs that preserve planarity.

Contents

A Simple Algorithm for r -gatherings on the Line	1
<i>Shin-ichi Nakano</i>	
Enumeration of Nonisomorphic Interval Graphs and Nonisomorphic Permutation Graphs	8
<i>Kazuaki Yamazaki, Toshiki Saitoh, Masashi Kiyomi, and Ryuhei Uehara</i>	
Secret Key Amplification from Uniformly Leaked Key Exchange Complete Graph	20
<i>Tatsuya Sasaki, Bateh Mathias Agbor, Shingo Masuda, Yu-ichi Hayashi, Takaaki Mizuki, and Hideaki Sone</i>	
Approximating Partially Bounded Degree Deletion on Directed Graphs	32
<i>Toshihiro Fujito, Kei Kimura, and Yuki Mizuno</i>	
Minimum-Width Annulus with Outliers: Circular, Square, and Rectangular Cases	44
<i>Hee-Kap Ahn, Taehoon Ahn, Sang Won Bae, Jongmin Choi, Mincheol Kim, Eunjin Oh, Chan-Su Shin, and Sang Duk Yoon</i>	
Minimum-Width Square Annulus Intersecting Polygons	56
<i>Hee-Kap Ahn, Taehoon Ahn, Jongmin Choi, Mincheol Kim, and Eunjin Oh</i>	
Two New Schemes in the Bitprobe Model	68
<i>Mirza Galib Anwarul Husain Baig and Deepanjan Kesh</i>	
Faster Network Algorithms Based on Graph Decomposition	80
<i>Manas Jyoti Kashyop, Tsunehiko Nagayama, and Kunihiko Sadakane</i>	
An Improvement of the Algorithm of Hertli for the Unique 3SAT Problem	93
<i>Tong Qin and Osamu Watanabe</i>	
Random Popular Matchings with Incomplete Preference Lists	106
<i>Suthee Ruangwises and Toshiya Itoh</i>	
Scheduling Batch Processing in Flexible Flowshop with Job Dependent Buffer Requirements: Lagrangian Relaxation Approach	119
<i>Hanyu Gu, Julia Memar, and Yakov Zinder</i>	
Computing Periods	132
<i>Junhee Cho, Sewon Park, and Martin Ziegler</i>	

A Note on Online Colouring Problems in Overlap Graphs
and Their Complements 144
Marc Demange and Martin Olsen

Online Facility Assignment 156
*Abu Reyan Ahmed, Md. Saidur Rahman,
and Stephen Kobourov*

Fault-Tolerant Complete Visibility for Asynchronous Robots
with Lights Under One-Axis Agreement 169
Aisha Aljohani, Pavan Poudel, and Gokarna Sharma

A Simple, Fast, Filter-Based Algorithm for Circular
Sequence Comparison 183
*Md. Aashikur Rahman Azim, Mohimenul Kabir,
and M. Sohel Rahman*

Boosting over Non-deterministic ZDDs 195
Takahiro Fujita, Kohei Hatano, and Eiji Takimoto

On Multiple Longest Common Subsequence and Common Motifs
with Gaps (Extended Abstract) 207
*Suri Dipannita Sayeed, M. Sohel Rahman,
and Atif Rahman*

FPT Algorithms Exploiting Carving Decomposition for Eulerian
Orientations and Ice-Type Models. 216
*Shinya Shiroshita, Tomoaki Ogasawara, Hidefumi Hiraishi,
and Hiroshi Imai*

On Structural Parameterizations of Happy Coloring, Empire Coloring
and Boxicity. 228
Jayesh Choudhari and I. Vinod Reddy

Complexity of the Maximum k -Path Vertex Cover Problem 240
*Eiji Miyano, Toshiki Saitoh, Ryuhei Uehara, Tsuyoshi Yagita,
and Tom C. van der Zanden*

On the Parallel Parameterized Complexity of the Graph
Isomorphism Problem 252
Bireswar Das, Murali Krishna Enduri, and I. Vinod Reddy

Author Index 265

A Simple Algorithm for r -gatherings on the Line

Shin-ichi Nakano(✉)

Gunma University, Kiryu 376-8515, Japan
nakano@cs.gunma-u.ac.jp

Abstract. In this paper we study recently proposed variant of the facility location problem called the r -gathering problem. Given sets C and F of points on the plane and distance $d(c, f)$ for each $c \in C$ and $f \in F$, an r -gathering of C to F is an assignment A of C to open facilities $F' \subset F$ such that r or more customers are assigned to each open facility. The cost of an r -gathering is the maximum distance $d(c, f)$ between $c \in C$ and $A(c) \in F'$ among the assignment, which is $\max_{c \in C} \{d(c, A(c))\}$. The r -gathering problem finds the r -gathering minimize the cost. A polynomial time 3-approximation algorithm for the r -gathering problem is known. When all C and F are on the line an $O((|C| + |F|) \log(|C| + |F|))$ time algorithm and an $O(|C| + |F| \log^2 r + |F| \log |F|)$ time algorithm to solve the r -gathering problem are known. In this paper we give a simple $O(|C| + r^2|F|)$ time algorithm to solve the r -gathering problem. Since in typical case $r \ll |F| \ll |C|$ holds our new algorithm is faster than the known algorithms.

Keywords: Algorithm · Facility location · Gathering

1 Introduction

The facility location problem and many of its variants are studied [D1, DH1]. In this paper we study a recently proposed variant of the problem, the r -gathering problem [AF1, A].

We start with a rather simpler problem. Given a set C of n points on the plane an r -gather-clustering is a partition of the points into clusters such that each cluster has at least r points. The cost of an r -gather-clustering is the maximum radius among the clusters, where the radius of a cluster is the minimum radius of the disk which can cover the points in the cluster. The r -gather-clustering problem [AF1] is the problem to find the r -gather-clustering minimizing the cost. The problem is NP-complete in general, however a polynomial time 2-approximation algorithm for the problem is known [AF1]. When all C are on the line, an $O(n \log n)$ time algorithm, based on the matrix search method [FJ1, AS1], for the problem is known [AN1].

In this paper we give an $O(rn)$ time simple algorithm to solve the problem when all C are on the line, by reducing the problem to the min-max path problem [GT1] in a weighted directed graph.

Assume that C is a set of residents on a street and we wish to locate emergency shelters for the residents so that each shelter serves r or more residents. Then r -gather clustering problem computes optimal locations for shelters which minimizing the evacuation time span, where each shelter for a cluster is located at the center of the cluster.

Then we consider the r -gathering problem. Given two sets C and F of points on the plane and distance $d(c, f)$ for each $c \in C$ and $f \in F$, an r -gathering of C to F is an assignment A of C to open facilities $F' \subset F$ such that r or more customers are assigned to each open facility. (Thus no customer is assigned to each facility in $F \setminus F'$.) The cost of an r -gathering is the maximum distance $d(c, f)$ between $c \in C$ and $A(c) \in F'$ among the assignment, which is $\max_{c \in C} \{d(c, A(c))\}$. The r -gathering problem finds the r -gathering minimizing the cost.

Armon [A] gave a simple 3-approximation algorithm for the r -gathering problem and proves that with the assumption $P \neq NP$ the problem cannot be approximated within a factor less than 3 for any $r \geq 3$. When all C and F are on the line an $O((|C| + |F|) \log(|C| + |F|))$ time algorithm [AN1] and an $O(|C| + |F| \log^2 r + |F| \log |F|)$ time algorithm [HN1] to solve the r -gathering problem are known.

In this paper we give an $O(|C| + r^2|F|)$ time algorithm to solve the r -gathering problem when all C and F are on the line. Since in typical case $r \ll |F| \ll |C|$ holds our new algorithm is faster than the known algorithms.

Assume that we are planning an evacuation plan for the residents on a street, F is a set of possible locations for emergency shelters, and $d(c, f)$ is the time needed for a person $c \in C$ to reach a shelter $f \in F$. Then an r -gathering (when all C and F are on the line) corresponds to an evacuation assignment such that each open shelter serves r or more people, and the r -gathering problem finds an evacuation plan minimizing the evacuation time span.

The remainder of this paper is organized as follows. In Sect. 2 we consider the r -gather-clustering problem and give an algorithm when all points in C are on the line. The idea of the algorithm is a reduction to the min-max path problem for a weighted directed graph. Then in Sect. 3 we give our algorithm for the r -gathering problem when all points in C and F are on the line. The idea of our algorithm is (1) a reduction to the min-max path problem for a weighted directed graph, and (2) carefully bounding the number of edges in the graph. Finally Sect. 4 is a conclusion.

2 r -gather-clustering on the Line

In this section we consider the r -gather-clustering problem, and give an algorithm when all points in C are on the line. Let $C = \{c_1, c_2, \dots, c_n\}$ be points on the horizontal line and we assume they are sorted from left to right. Our idea is to reduce the r -gather-clustering problem to the mix-max path problem in a weighted directed (acyclic) graph. First we have the following two lemmas.

Lemma 1. *There exists an optimal solution in which the points in each cluster are consecutive in C .*

Proof. Assume otherwise. We say three points $c_a, c_b, c_c \in C$ with (1) $a < b < c$, (2) $c_a, c_c \in C_x$ and (3) $c_b \in C_y$ where C_x and C_y are clusters are *crossing triple*. By the assumption above any optimal solution has some crossing triple. Let S be the solution of the r -gather-clustering problem (when all points in C are on the line) with the minimum number of crossing triples. Let S' be the r -gather clustering derived from S by replacing C_x and C_y by C'_x and C'_y so that C'_x is the leftmost $|C_x|$ points in $C_x \cup C_y$ and C'_y is the rightmost $|C_y|$ points in $C_x \cup C_y$. Now the cost of S' is smaller than the cost of S , or S' has less crossing triples. A contradiction. \square

Thus we can assume each cluster in an optimal solution consists of consecutive points $\{c_i, c_{i+1}, \dots, c_j\}$ for some i and j .

Lemma 2. *There exists an optimal solution in which the number of points in each cluster is at most $2r - 1$.*

Proof. Assume otherwise. Then optimal solution has a cluster with more than $2r$ points. Then divide each of such clusters into two (or more) clusters, respectively, so that each cluster has r or more points, but at most $2r - 1$ points. Since this modification does not increase the cost, the resulting clustering is also a solution. \square

Then we define the directed (acyclic) graph $D(V, E)$ and the weight of each edge, as follows.

$$V = \{p_0, p_1, p_2, \dots, p_n\}$$

$$E = \{(p_i, p_j) | i + r \leq j \leq i + 2r - 1\}$$

See an example with $r = 3$ in Fig. 1. Note that the number $|E|$ of edges is at most rn . The weight of an edge (p_i, p_j) is the half of the distance between c_{i+1} and c_j , and denoted by $w(p_i, p_j)$.

The cost of a directed path from p_0 to p_n is defined by the weight of the edge having the maximum weight in the directed path. *The min-max path* from p_0 to p_n is the directed path from p_0 to p_n with the minimum cost.

Now C has an r -gather-clustering with cost k iff $D(V, E)$ has a directed path from p_0 to p_n with cost k . See Fig. 2.

Thus if we can compute the min-max path in D then it corresponds to the solution of the r -gather-clustering problem. Intuitively, each (directed) edge in the min-max path corresponds to a cluster of an r -gather-clustering.

We can construct the $D(V, E)$ in $O(rn)$ time. An $O(|E| \log^* |V|)$ time algorithm for the min-max path problem for a directed graph $D = (V, E)$ is known [GT1]. However, since $D(V, E)$ is a DAG (directed acyclic graph) we can compute the min-max path from p_0 to p_n in $O(|E|)$ time by a simple dynamic programming algorithm. (Let w_i be the cost of the min-max path from p_0 to p_i . For each

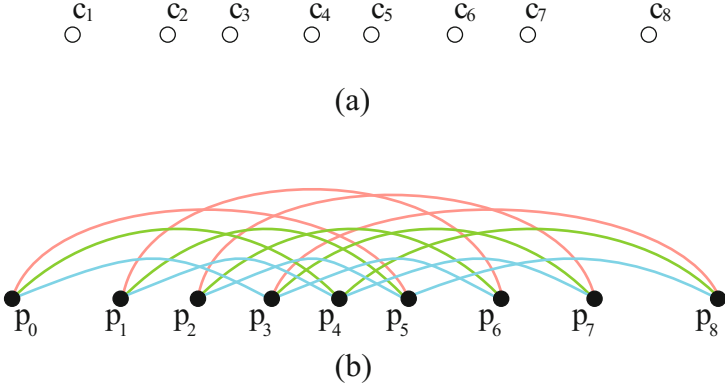


Fig. 1. (a) A point set C and (b) the weighted directed graph D with $r = 3$.

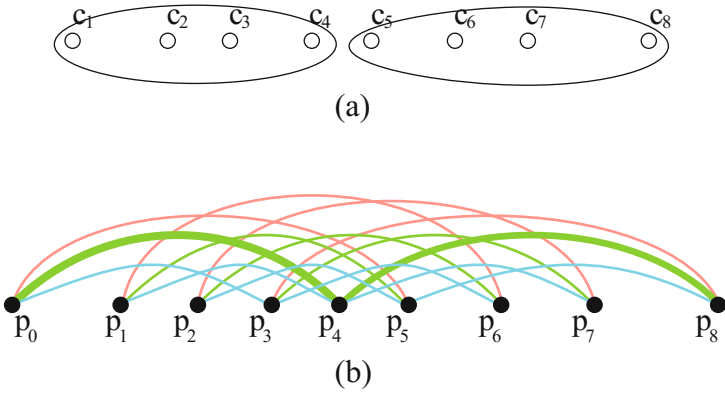


Fig. 2. (a) an r -gather clustering (b) its corresponding min-max path of D .

p_i we can compute w_i by checking each incoming edge (p_x, p_i) to p_i and the cost w_x of the min-max path from p_0 to p_x .)

Thus we have the following theorem.

Theorem 1. *One can solve the r -gather-clustering problem in $O(rn)$ time, when all points in C are on the line.*

3 r -gathering

In this section we give an algorithm for the r -gathering problem when all points in C and F are on the line, by reducing the problem to the min-max path problem for a weighted directed graph, and bounding the number of edges in the graph.

Let $C = \{c_1, c_2, \dots, c_n\}$ and $F = \{f_1, f_2, \dots, f_m\}$ be points on the horizontal line and we assume they are sorted from left to right, respectively. Similar to Lemma 1 we can assume the points assigned to a facility are consecutive in a solution.

For consecutive three facilities f_{j-1} , f_j and f_{j+1} in F let m_L be the midpoint of f_{j-1} and f_j , and m_R the midpoint of f_j and f_{j+1} . We have the following two lemma.

Lemma 3. *Assume that C has $2r$ or more points on the left of m_L . Let c_i be the $2r$ -th point from right in C' where C' is the set of points in C on or left of m_L . There exists an optimal solution in which $c_{i'}$ with $i' < i$ is never assigned to f_j .*

Proof. Assume for a contradiction such $c_{i'}$ is assigned to f_j . We have two cases.

If the rightmost point assigned to f_j is on the left of m_L then reassigning the points assigned to f_j to f_{j-1} results in a new r -gathering and since it does not increase the cost the resulting r -gathering is also a solution of the given r -gathering problem.

Otherwise, the rightmost point assigned to f_j is on or right of m_L . Then at least $2r$ points on or left of m_L are assigned to f_j by Lemma 1, with other points on the right of m_L . Let C' be the subset of C consisting of the points (1) assigned to f_j , (2) on or left of m_L , and (3) but not the rightmost r points on or left of m_L . Note that $|C'| \geq r$ holds and C' contains $c_{i'}$. Reassigning the points in C' to f_{j-1} results in a new r -gathering and the resulting r -gathering is also a solution since it does not increase the cost. \square

Intuitively if $c_{i'}$ is too far from f_j then $c_{i'}$ is never assigned to f_j . Symmetrically we have the following lemma.

Lemma 4. *If C has $2r$ or more points on the right of m_R , then $c_{i'}$ with $i' > i$ is never assigned to f_j , where c_i is the $2r$ -th point in C on or right of m_R .*

We have more lemmas. Let C' be the set of points between m_L and m_R except the leftmost $2r$ points and the rightmost $2r$ points.

Lemma 5. *If C has $5r$ or more points between m_L and m_R , then the customers in C' are assigned to f_j in a solution of the r -gathering problem.*

Proof. Immediate from the two lemmas above. \square

Thus if we can compute the solution for $C - C'$ then appending the assignment from the points in C' to f_j results in the solution for C . From now on we assume we have removed every such C' from C .

We have more lemmas for the boundary case. Let m be the midpoints of f_1 and f_2 in F .

Lemma 6. *If C has $2r$ or more points on the left of m , then each $c_{i'}$ with $i' < i$ is assigned to f_1 in a solution of the r -gathering problem, where c_i is the $2r$ -th customer in C on the left of m .*

Proof. Immediate from Lemma 3. \square

Let m be the midpoints of f_{m-1} and f_m in F .

Lemma 7. *If C has $2r$ or more points on the right of m , then each $c_{i'}$ with $i' > i$ is assigned to f_m in a solution of the r -gathering problem, where c_i is the $2r$ -th customer in C on the right of m .*

Thus we have the following lemma.

Lemma 8. *The number of points in C possibly assigning to each facility $f \in F$ is at most $9r$.*

Proof. For each f_j with $1 < j < m$ define m_L and m_R as above. The number of points possibly assigning to f_j is (1) at most $2r$ on the left of m_L , (2) at most $2r$ on the right of m_R , and (3) at most $5r$ between m_L and m_R , by the lemmas above. Similar for f_1 and f_m . \square

Now we are going to define a weighted directed graph $D(V, E)$ for F and C , and the weight of each edge.

The set of vertices is defined as follows.

$$V = \{p_0, p_1, p_2, \dots, p_n\}$$

For each facility f_h with $h = 2, 3, \dots, m-1$ and its possible cluster consisting of points $\{c_{i+1}, c_{i+2}, \dots, c_j\}$ we define an edge (p_i, p_j) . So (p_i, p_j) is an edge iff

- (1) $i + r \leq j \leq i + 2r - 1$
- (2) $i \geq i'$ where $c_{i'}$ is the $2r$ -th customer on the left of m_L , and
- (3) $j \leq j'$ where $c_{j'}$ is the $2r$ -th customer on the right of m_R ,

where m_L and m_R are defined for f_h as above. Let E_h be the set of edges consisting of edges defined for f_h above. Similarly we define E_1 and E_m .

Finally,

$$E = E_1 \cup E_2 \cup \dots \cup E_m$$

Note that E may contain many multi-edges.

The weight of an edge (p_i, p_j) for f_h is the maximum of (1) the distance between c_{i+1} and f_h , and (2) the distance between c_j and f_h .

The cost of a directed path from p_0 to p_n is defined by the weight of the edge having the maximum weight in the directed path. *The min-max path* from p_0 to p_n is the directed path from p_0 to p_n with the minimum cost.

We need to compute for each f_h the $2r$ -th customer on the left of m_L and the $2r$ -th customer on the right of m_R . By scanning the line we can compute them for all f_h in $O(|F| + |C|)$ time in total. Note that each edge in E corresponds to a pair of customers possibly assigning to a common facility. Thus the number of the edges in E is at most $81r^2|F|$ by Lemma 8. Thus we can construct $D(V, E)$ in $O(|F| + |C| + r^2|F|)$ time in total.

Similar to Sect. 2 we have reduced the r -gathering problem to the min-max path problem, and have the following theorem.

Theorem 2. *When both C and F are on the line one can solve the r -gathering problem in $O(n + r^2m)$ time, where $n = |C|$ and $m = |F|$.*

4 Conclusion

In this paper we have presented an algorithm to solve the r -gather clustering problem when all C are on the line. The running time of the algorithm is $O(rn)$, where $n = |C|$. We also presented an algorithm to solve the r -gathering problem, which runs in time $O(n + r^2m)$, where $n = |C|$ and $m = |F| < n$.

Can we solve the problem more efficiently or for more general input or cost?

References

- [AF1] Aggarwal, G., Feder, T., Kenthapadi, K., Khuller, S., Panigrahy, R., Thomas, D., Zhu, A.: Achieving anonymity via clustering. *Trans. Algorithms* **6** (2010). Article No. 49
- [AS1] Agarwal, P., Sharir, M.: Efficient algorithms for geometric optimization. *Comput. Surv.* **30**, 412–458 (1998)
- [AN1] Akagi, T., Nakano, S.: On r -gatherings on the line. In: Wang, J., Yap, C. (eds.) FAW 2015. LNCS, vol. 9130, pp. 25–32. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19647-3_3
- [A] Armon, A.: On min-max r -gatherings. *Theor. Comput. Sci.* **412**, 573–582 (2011)
- [D1] Drezner, Z.: *Facility Location: A Survey of Applications and Methods*. Springer, New York (1995)
- [DH1] Drezner, Z., Hamacher, H.W.: *Facility Location: Applications and Theory*. Springer, New York (2004)
- [FJ1] Frederickson, G., Johnson, D.: Generalized selection and ranking: sorted matrices. *SIAM J. Comput.* **13**, 14–30 (1984)
- [GT1] Gabow, H., Tarjan, R.: Algorithms for two bottleneck optimization problems. *J. Algorithms* **9**, 411–417 (1988)
- [HN1] Han, Y., Nakano, S.: On r -gatherings on the line. In: *Proceedings of FCS 2016*, pp. 99–104 (2016)

Enumeration of Nonisomorphic Interval Graphs and Nonisomorphic Permutation Graphs

Kazuaki Yamazaki¹, Toshiki Saitoh², Masashi Kiyomi³, and Ryuhei Uehara¹(✉)

¹ School of Information Science, Japan Advanced Institute of Science and Technology (JAIST), Nomi, Japan
`{torus711,uehara}@jaist.ac.jp`

² School of Computer Science and Systems Engineering, Kyushu Institute of Technology, Kitakyushu, Japan
`toshikis@ces.kyutech.ac.jp`

³ International College of Arts and Sciences, Yokohama City University, Yokohama, Japan
`masashi@yokohama-cu.ac.jp`

Abstract. In this paper, a general framework for enumerating every element in a graph class is given. The main feature of this framework is that it is designed to enumerate only non-isomorphic graphs in a graph class. Applying this framework to the classes of interval graphs and permutation graphs, we give efficient enumeration algorithms for these graph classes such that each element in the class is output in a polynomial time delay. The experimental results are also provided. The catalogs of graphs in these graph classes are also provided.

1 Introduction

Recently we have to process huge amounts of data in the area of data mining, bioinformatics, etc. In most cases, we have to use some certain structure to solve problems efficiently. We need three efficiencies to deal with a complex structure; it has to be represented efficiently, essentially different instances have to be enumerated efficiently, and its properties have to be checked efficiently. From the viewpoint of the “difference,” in graphs, it is natural to consider that two graphs are different when they are non-isomorphic. However, in general, the graph isomorphism problem is difficult to solve efficiently even on restricted graph classes (see [19]). Though, there are rich structures even if we restricted to the graph classes that allow us to solve graph isomorphism efficiently.

We investigate the enumeration of a graph class from this viewpoint in this paper. In this context, there are two previous results by some of the authors [16,17]. In the paper, the authors gave efficient enumeration algorithms for proper interval graphs and bipartite permutation graphs. However, they are quite specific to some common properties of these graph classes, and it is unlikely to extend to other graph classes. Therefore, we focus on some graph classes such that graph isomorphism can be solved efficiently, and we develop a general framework that gives us to enumerate all non-isomorphic graphs with n vertices for

a given integer n , in each of these graph classes. Intuitively, most of the graph classes in which graph isomorphism can be solved in polynomial time share a common property: Each graph in the graph class can be characterized by a canonical tree structure, and graph isomorphism can be checked essentially by solving the graph isomorphism problem on these labeled trees [12].

There are two well-known graph classes that graph isomorphism can be solved in polynomial time in this manner; interval graphs [14] and permutation graphs [4]. We mention that these graph classes have been widely investigated since they have many applications, and they are very basic graph classes from the viewpoints of graph theory and algorithms. Therefore many useful properties have been revealed, and many efficient algorithms have been developed for them (see, e.g., [3, 7, 18]). From the practical point of view, when an efficient algorithm for a graph class is developed and implemented, we need many graphs belonging to the class to check the reliability of the algorithm. Thus, for such popular graph classes, efficient enumerations are required [9]. However, as far as the authors know, these concrete catalogs for these graph classes have never been provided.

In this paper, we first propose a general framework of enumeration of a graph class in which graph isomorphism can be solved in polynomial time. Then we turn to the details of applications of this framework to interval graphs and permutation graphs. We finally give the experimental results of the implementations for these graph classes. That is, we give the first actual catalogs of non-isomorphic graphs for these graph classes for small n , where n is the number of vertices. (We note that, for interval graphs, some related results can be found in [8] from the viewpoint of counting, not enumeration.) Due to space limitation, all proofs and some figures are omitted.

2 Preliminaries

We only consider simple graph $G = (V, E)$ with no self-loop and multiple edges. We assume $V = \{1, 2, \dots, n\}$ for some n , and $|E| = m$. For two integers i, j , we denote by $G + \{i, j\}$ the graph $(V, E \cup \{\{i, j\}\})$, and by $G - \{i, j\}$ the graph $(V, E \setminus \{\{i, j\}\})$. Let K_n denote the complete graph of n vertices and P_n denote the path of n vertices of length $n - 1$.

A graph (V, E) with $V = \{1, 2, \dots, n\}$ is an *interval graph* when there is a finite set of intervals $\mathcal{I} = \{I_1, I_2, \dots, I_n\}$ on the real line such that $\{i, j\} \in E$ if and only if $I_i \cap I_j \neq \emptyset$ for each i and j with $0 < i, j \leq n$. We call the interval set \mathcal{I} an *interval representation* of the graph. For each interval I , we denote by $L(I)$ and $R(I)$ the left and right endpoints of the interval, respectively (hence we have $L(I) \leq R(I)$ and $I = [L(I), R(I)]$).

A graph $G = (V, E)$ with $V = \{1, 2, \dots, n\}$ is a *permutation graph* when there is a permutation π over V such that $\{i, j\} \in E$ if and only if $(i - j)(\pi(i) - \pi(j)) < 0$. Intuitively, each vertex i in a permutation graph corresponds to a line ℓ_i joining two points on two parallel lines L_1 and L_2 such that two vertices i and j are adjacent if and only if the corresponding lines ℓ_i and ℓ_j intersect. We suppose that the indices $1, 2, \dots, n$ of the vertices give the ordering of the points on L_1 ,

and the ordering by permutation π over V gives the ordering of the points on L_2 . That is, ℓ_i joins the i th vertex on L_1 and the $\pi(i)$ th vertex on L_2 . We call this intersection model a *line representation* of the permutation graph.

We define a *graph isomorphism* between two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ as follows. The graph G_1 is isomorphic to G_2 when there is a one-to-one mapping $\phi : V_1 \rightarrow V_2$ such that for any pair of vertices $u, v \in V_1$, $\{u, v\} \in E_1$ if and only if $\{\phi(u), \phi(v)\} \in E_2$. We denote by $G_1 \sim G_2$ for two isomorphic graphs G_1 and G_2 .

3 General Framework

For a graph class \mathcal{C} , we suppose that the graph isomorphism can be solved in polynomial time for \mathcal{C} . We denote by $\text{Iso}(n)$ the time complexity for solving the graph isomorphism problem for two graphs G_1 and G_2 of n vertices in the class \mathcal{C} . Here we define the notion of the *canonical* graph for any given graph G in \mathcal{C} with respect to the graph isomorphism. We first suppose that we can define a transitive ordering $<$ over isomorphic graphs in \mathcal{C} . That is, (1) either $G_1 < G_2$ or $G_2 < G_1$ holds for any given two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ such that $G_1 \sim G_2$ and $E_1 \neq E_2$, and (2) when $G_1 < G_2$ and $G_2 < G_3$ for three isomorphic graphs G_1, G_2, G_3 , we have $G_1 < G_3$. Then there exists a unique *minimal* graph G for any set of all isomorphic graphs in \mathcal{C} . We call this graph G the *canonical* graph. Our goal is to enumerate all canonical graphs in the class \mathcal{C} . To this goal, we will use the following properties of the class \mathcal{C} :

Canonical property: For any graph G in \mathcal{C} , we can compute its canonical graph in polynomial time. That is, the canonical property guarantees that any graph G can be dealt with its canonical form (in polynomial time).

We use *reverse search* technique to enumerate all graphs (see [1] for the details about reverse search). In reverse search, we define a family tree \mathcal{T} over the graphs in the target graph class \mathcal{C} by introducing a parent-child relationship between two graphs G and G' in \mathcal{C} . More precisely, in the class \mathcal{C} , we first fix a root node¹ $G_R \in \mathcal{C}$. In this paper, we will use K_n as the root node G_R , since K_n belongs to interval graphs and permutation graphs. For each graph $G \in \mathcal{C} \setminus \{G_R\}$, we assume that its *parent* G' of G is uniquely defined and computed in polynomial time. We will define the parent-child relationship so that it is acyclic, it forms a tree on the graphs rooted at G_R in \mathcal{C} . Thus we call the resulting tree spanning the class \mathcal{C} *family tree*, and denoted by \mathcal{T} .

For the current graph G , we will modify G by some *basic operation* to find its parent or children in \mathcal{T} of the class \mathcal{C} . In this paper, we will use “add an edge” as a basic operation to find its parent. The key requirement is that the parent should be uniquely determined for each graph except the root node in \mathcal{T} .

¹ We use two terms “node” and “vertex” to indicate an element in a graph. When we use “vertex,” it indicates a vertex in the original graph G in the class \mathcal{C} . On the other hand, when we use “node,” it indicates meta-structure of graphs. That is, a “node” in \mathcal{T} indicates a graph in the class \mathcal{C} .

In an interval graph (or in a permutation graph) G which is not K_n , there is at least one edge e such that $G + e$ is an interval graph (or a permutation graph, respectively). When there are two or more such edges e , we have to determine the unique parent efficiently. To determine the unique parent for any given graph $G \in \mathcal{C} \setminus \{G_R\}$, we need the following *operational property*:

Operational property: Let G be any graph in $\mathcal{C} \setminus \{G_R\}$, where G_R is the root node of \mathcal{T} of \mathcal{C} . Then, there exists at least one graph $G' \in \mathcal{C}$ such that G' is obtained from G by applying one basic operation. Moreover, we can find minimal G' , which is determined uniquely, among them in polynomial time.

The operational property guarantees that we can find a unique parent of G for a given graph G in $\mathcal{C} \setminus \{G_R\}$ in polynomial time. However, in reverse search, a graph G produces the set of potential *children* $S(G)$. Precisely, the algorithm first produces a set of graphs $S'(G)$ that consists of the graphs obtained by applying the reverse of basic operation. In our context, $S'(G)$ is the set of graph $G - e$ for each edge e . It is guaranteed that all children in the family tree are in $S'(G)$, but there may be redundant graphs. There are three considerable cases. The first case is easy; when $G - e$ is not in \mathcal{C} , just discard it. The second case is that G produces two or more isomorphic graphs by the reverse of basic operation. For example, when G is a complete graph and the basic operation is “add an edge,” G produces all graphs $G - \{i, j\}$ for all $1 \leq i, j \leq n$ as potential children of G . In this case, the algorithm discards all isomorphic graphs except one. Let $S(G)$ be the set of the nonisomorphic graphs in \mathcal{C} obtained from G by the reverse of basic operation. The last considerable case is that the graph $G' \in S(G)$ has a different parent. This case occurs when G' has two (or more) edges e_1 and e_2 such that $G' + e_1 \in \mathcal{C}$ and $G' + e_2 \in \mathcal{C}$. In this case, G' appears in both of $S(G' + e_1)$ and $S(G' + e_2)$. To avoid redundancy, G' will check which is the unique parent.

Now we are ready to show the outline of the enumeration algorithm:

Algorithm 1. Outline of Enumeration Algorithm based on Reverse Search

Input : An integer n

Output: All nonisomorphic graphs of n vertices in a graph class \mathcal{C}

A set \mathcal{S} is initialized by the root node of the family tree of \mathcal{C} ;

while \mathcal{S} is not empty **do**

Pick up one node that represents a graph $G = (V, E)$ in the class \mathcal{C} ;

Output G as an element in the class \mathcal{C} ;

Compute the set $S(G)$ of nonisomorphic graphs in \mathcal{C} obtained by the reverse of basic operation;

// G may produce two or more isomorphic graphs, which should be avoided here.

foreach $G' \in S(G)$ **do**

// Check if G is the unique parent of G' .

Compute the unique parent \hat{G}' of G' ;

If \hat{G}' is isomorphic to G , push G' into \mathcal{S} ;

The algorithm enumerates all elements in breadth first search (BFS) manner when \mathcal{S} is realized by a queue, and in depth first search (DFS) manner when \mathcal{S} is realized by a stack. Hereafter, we suppose that it runs in BFS, which makes proof of correctness simpler.

Let \mathcal{C} be the graph class satisfying the properties above. Then we have the main theorem for the framework:

Theorem 1. *We can enumerate all nonisomorphic graphs of n vertices in \mathcal{C} with polynomial time delay. That is, the running time of the algorithm is $|\mathcal{C}_n|p(n)$ for some polynomial function p , where \mathcal{C}_n denotes the subset of \mathcal{C} that contains all graphs of n vertices in \mathcal{C} .*

By Theorem 1, we can establish that there are several graph classes that admit to enumerate all elements in the class in polynomial time delay. However, the efficiency of the enumeration is strongly depending on the detailed implementation for each class. We show two efficient implementations for two representative graph classes; interval graphs and permutation graphs. We also show experimental results, that is, we give catalogs for these graph classes. In both of interval graphs and permutation graphs, we let K_n be the root node of the family tree, and basic operation is “add an edge” to obtain the parent.

4 Enumeration of Nonisomorphic Interval Graphs

We first focus on the enumeration of interval graphs of n vertices. Let \mathcal{C} be the set of interval graphs of n vertices in this section. We first show the operational property for $\mathcal{C} \setminus \{G_R\}$, where $G_R \sim K_n$. (We note that K_n is not only an interval graph, but also a permutation graph, and we use it as a common root node of the family trees for both graph classes.)

Lemma 1 [11]. *Let $G = (V, E)$ be any interval graph which is not K_n . Then G has at least one edge e such that $G + e$ is also an interval graph.*

4.1 Canonical Representation

We turn to the canonical representation of an interval graph. We first show the canonical tree structure, and then we give how to obtain a canonical representation for the graph.

Canonical Tree Representation. As the tree structure for an interval graph, we use the \mathcal{MPQ} -tree model. The notion was developed by Korte and Möhring [13] as a kind of labeled \mathcal{PQ} -tree introduced by Booth and Lueker [2]. We here give a brief idea, and the details can be found in journal version.

A \mathcal{PQ} -tree is a rooted tree T^* with two types of internal nodes: \mathcal{P} and \mathcal{Q} , which will be represented by circles and rectangles, respectively. The leaves of T^* are labeled 1-1 with the maximal cliques of the interval graph G . The *frontier* of a \mathcal{PQ} -tree T^* is the permutation of the maximal cliques obtained by the

ordering of the leaves of \mathcal{T}^* from left to right. Two \mathcal{PQ} -trees \mathcal{T}^* and \mathcal{T}'^* are *equivalent*, if one can be obtained from the other by applying the following rules; (1) arbitrarily permute the child nodes of a \mathcal{P} -node, or (2) reverse the order of the child nodes of a \mathcal{Q} -node. A graph G is an interval graph if and only if there is a \mathcal{PQ} -tree \mathcal{T}^* whose frontier represents a consecutive arrangement of the maximal cliques of G . The \mathcal{MPQ} -tree \mathcal{T} assigns sets of vertices (possibly empty) to the nodes of a \mathcal{PQ} -tree \mathcal{T}^* representing an interval graph. A \mathcal{P} -node is assigned only one set, while a \mathcal{Q} -node has a set for each of its children (ordered from left to right according to the ordering of the children).

For a \mathcal{P} -node P , this set consists of those vertices of G contained in all maximal cliques represented by the subtree of P in \mathcal{T} , but in no other cliques.

For a \mathcal{Q} -node Q , the definition is more involved. Let Q_1, \dots, Q_m be the set of the children (in consecutive order) of Q , and let \mathcal{T}_i be the subtree of \mathcal{T} with root Q_i . We then assign a set S_i , called *section*, to each Q_i . Section S_i contains all vertices that are contained in all maximal cliques of \mathcal{T}_i and some other \mathcal{T}_j , but not in any clique belonging to some other subtree of \mathcal{T} that is not below Q . The key property of \mathcal{MPQ} -trees is summarized as follows:

Theorem 2 [13, Theorem 2.1]. *Let \mathcal{T} be the \mathcal{MPQ} -tree for an interval graph $G = (V, E)$. Then we have the following: (a) \mathcal{T} can be computed in linear time and space. (b) Each maximal clique of G corresponds to a path in \mathcal{T} from the root to a leaf, where each vertex $v \in V$ is as close as possible to the root. (c) In \mathcal{T} , each vertex v appears in either one leaf, one \mathcal{P} -node, or consecutive sections $S_i, S_{i+1}, \dots, S_{i+j}$ for some \mathcal{Q} -node with $j > 0$.*

For two interval graphs G_1 and G_2 , let \mathcal{T}_1 and \mathcal{T}_2 be the corresponding \mathcal{MPQ} -trees. Then $G_1 \sim G_2$ if and only if $\mathcal{T}_1 \sim \mathcal{T}_2$ (as labeled trees).

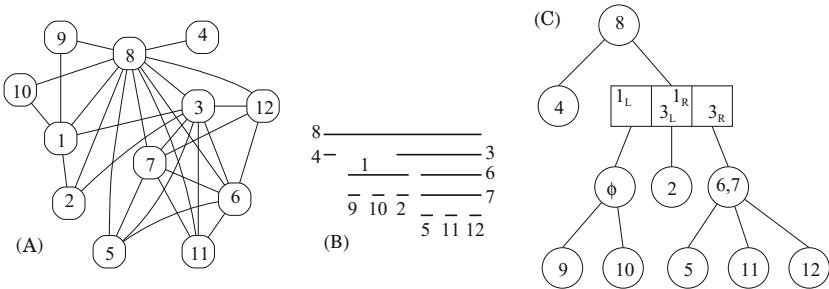


Fig. 1. An interval graph, its interval representation, and its corresponding \mathcal{MPQ} -tree.

A simple example is given in Fig. 1. For a given interval graph G in Fig. 1(A), its interval representation is given in Fig. 1(B), and the corresponding \mathcal{MPQ} -tree is given in Fig. 1(C).

Canonical String Representation. The \mathcal{MPQ} -tree \mathcal{T} for an interval graph $G = (V, E)$ is the canonical form in the sense that for any two isomorphic interval graphs $G_1 \sim G_2$, the resulting \mathcal{MPQ} trees \mathcal{T}_1 and \mathcal{T}_2 are also isomorphic and they can be used to solve the graph isomorphism problem for G_1 and G_2 in linear time since it can be solved in linear time on such labeled trees. We further introduce a *canonical string representation* for a given interval graph to decide the parent of an interval graph uniquely. Intuitively, we will introduce a string representation for an interval graph so that if two interval graphs are isomorphic, their corresponding strings are exactly the same. We here define two basic cases: a complete graph K_n is represented by $1234 \dots (n-1)nn(n-1) \dots 4321$ and a path P_n is represented by $1213243 \dots i(i-1)(i+1)i \dots n(n-1)n$. To define general canonical string representations, we need more details. The translation from a given \mathcal{MPQ} -tree to the canonical string consists of three phases.

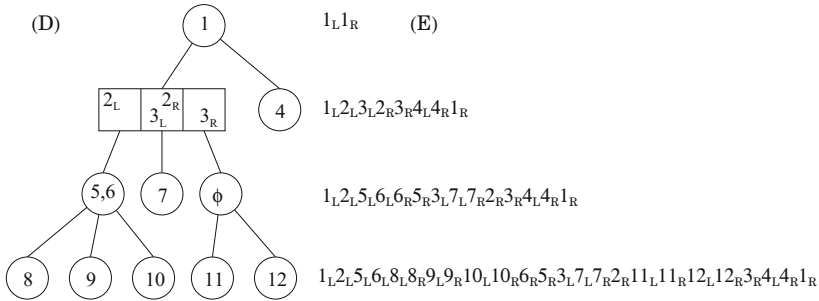


Fig. 2. The \mathcal{MPQ} -tree in left-to-right ordering with relabeling, and its canonical string.

First, we draw the \mathcal{MPQ} -tree as an ordered tree which is a rooted tree with left-to-right ordering specified by the children of each node. The children for a node are arranged in the ordering from “left-heavy” to “right-light.” That is, we introduce a total ordering over all \mathcal{MPQ} -trees that is a transitive relationship. This idea can be found in [10], and the details of the ordering for an \mathcal{MPQ} -tree is omitted here. The key property of the ordering is that $\text{Ind}(\mathcal{T}_1)$ and $\text{Ind}(\mathcal{T}_2)$ for two \mathcal{MPQ} -trees are equal if and only if they are isomorphic. Once we draw the \mathcal{MPQ} -tree in the way of the ordered tree defined by the ordering, two drawings of \mathcal{T}_1 and \mathcal{T}_2 are the same (except vertex labelings) if and only if they are isomorphic.

In the second phase, we relabel the vertices $V = \{1, 2, 3, \dots, n\}$ according to the ordering in the breadth first search manner on the drawing of the tree. (We suppose that a left node is visited before a right node.) By this traverse of vertices of V in the nodes in a \mathcal{MPQ} -tree with the basic rule that the canonical representation of K_n is $1234 \dots (n-1)nn(n-1) \dots 4321$, we can observe that two \mathcal{MPQ} -trees \mathcal{T}_1 and \mathcal{T}_2 are isomorphic if and only if the resulting drawings are completely the same including the labels of vertices in V . In this sense, we call the relabeled \mathcal{MPQ} -tree \mathcal{T} for an interval graph G the *canonical \mathcal{MPQ} -tree*

of G . For example, when we apply this process to the \mathcal{MPQ} -tree in Fig. 1(C), we obtain the canonicalized \mathcal{MPQ} -tree in Fig. 2(D).

In the last phase, we again traverse this canonical \mathcal{MPQ} -tree \mathcal{T} in breadth first search manner and generate the canonical string of \mathcal{T} as follows: for a \mathcal{P} -node, the algorithm first outputs all left endpoints of the vertices in the node, make recursive calls for each of its children, and output all right endpoints of the vertices in the node following the basic rule of K_n . For a \mathcal{Q} -node, the algorithm processes each section by section in the \mathcal{Q} -node. Let $\text{Str}_I(G)$ be resulting string representation for a given interval graph G . From the canonicalized \mathcal{MPQ} -tree in Fig. 2(D), we obtain the canonical string “1 2 5 6 8 8 9 9 10 10 6 5 3 7 7 2 11 11 12 12 3 4 4 1.” In Fig. 2(E), we add L and R that indicate left and right endpoints, respectively. We also give each corresponding string for each subtree rooted at the original root up to level 0, 1, 2, and 3. Combining the results in [13] and definitions above, we obtain the following theorem.

Theorem 3. *Let $G = (V, E)$ be any interval graph with $|V| = n$ and $|E| = m$. (1) The canonical \mathcal{MPQ} -tree of G and $\text{Str}_I(G)$ can be computed in $O(n + m)$ time. (2) $|\text{Str}_I(G)| = 2n$. (3) Two interval graphs G_1 and G_2 are isomorphic if and only if $\text{Str}_I(G_1) = \text{Str}_I(G_2)$.*

4.2 Parent-Child Relationship

As shown in Lemma 1, for any given interval graph $G = (V, E)$ with $G \not\sim K_n$, there is at least one edge $e = \{u, v\}$ with $e \notin E$ such that $G + e$ is also an interval graph. For the graph G , let \mathcal{T} be the canonical \mathcal{MPQ} -tree of G . Without loss of generality, we assume that \mathcal{T} is consistent to G from the viewpoint of labels. That is, when we make \mathcal{T} from G , the relabeling process does not change any label of a vertex in V . By Theorem 3, these G and \mathcal{T} can be obtained in linear time. Now we let $\hat{E} = \{e = \{u, v\} \mid G + e \text{ is an interval graph}\}$. Among \hat{E} , we can pick up a unique edge \hat{e} that is the lexicographically smallest element in \hat{E} . We define the parent of G by $G + \hat{e}$. Clearly, the parent is uniquely determined.

Theorem 4. *Let $G = (V, E)$ be any interval graph with $|V| = n$ and $|E| = m$. Then its parent can be computed in $O(n^2(n + m))$ time.*

4.3 Algorithm Analysis

We here analyze the algorithm and show that each graph is enumerated in polynomial time, which guarantees that this algorithm achieves the polynomial time delay for each graph. The root node can be enumerated in polynomial time since it contains K_n . For each graph G in \mathcal{C} , we evaluate its running cost consists of its output, the computation of $S(G)$, and the process for each $G' \in S(G)$. The output of G takes $O(n + m)$ time. In this framework with the basic operation, the set $S(G)$ contains at most m children, each of which is obtained from G by removing an edge. It takes $O(m(n + m))$ time (by maintaining the set of canonical string representations in a reasonable data structure, e.g., trie (or prefix tree), we

can reduce isomorphic graphs in this process). Then we obtain the set of $O(m)$ graphs, and each G' of them has n vertices and $m - 1$ edges. Now the algorithm checks if the unique parent of each G' is G or not. It takes $O(n^2(n + m))$ time by Theorem 4 for each. Thus, this process takes $O(n^2m(n + m))$ time in total. Therefore, each graph consumes $O(n^2m(n + m))$ time in total when it is output. Since $m = O(n^2)$ in general, our enumeration algorithm runs in $O(n^6)$ time per graph.

Our main theorem in this section is summarized as follows:

Theorem 5. *We can enumerate every nonisomorphic interval graph of n vertices. Each interval graph is enumerated in $O(n^6)$ time delay.*

4.4 Three Variants of Enumeration

Corollary 1. *The algorithm in Theorem 5 can be modified to enumerate (1) connected graphs, and/or (2) at most n vertices. In any variant, the delay is not changed from $O(n'^6)$, where n' is the number of vertices of the output graph.*

5 Enumeration of Nonisomorphic Permutation Graphs

We next focus on the enumeration of permutation graphs of n vertices. Let \mathcal{C} be the set of permutation graphs of n vertices in this section. We first show the operational property for $\mathcal{C} \setminus \{K_n\}$, where $G_R \sim K_n$.

Lemma 2. *Let $G = (V, E)$ be any permutation graph which is not K_n . Then G has at least one edge e such that $G + e$ is also a permutation graph.*

5.1 Canonical Representation

Now we turn to the canonical representation of permutation graphs. First, we introduce the notion of *modular decomposition tree*.

Canonical Tree Representation. For a graph $G = (V, E)$, a vertex set $X \subseteq V$ is a *module* if and only if every vertex x not in X , either every member of X is adjacent to x or no member of X is adjacent to x . (See [15] for the details.) *Trivial modules* are \emptyset , V , and all the singletons $\{v\}$ for $v \in V$. A graph (or a module) is *prime* if and only if all its modules are trivial. For any permutation graph G , G has a unique line representation (up to reversal) if and only if it is prime [6].

In [6], Gallai defined the *modular decomposition* recursively on a graph with vertex set V . Intuitively, maximal modules give a unique partition of V recursively, and we have a tree structure with respect to the partition, which is called the *modular decomposition tree*. In a modular decomposition tree \mathcal{T} , if all children are joined by edges in the original graph, the parent of them is called *series* node, and if all children are independent, the parent is called *parallel* node.

It is well known that the modular decomposition tree for a permutation graph (1) is canonical up to isomorphism [4], and (2) can be computed in linear time and space (see, e.g., [5]).

In out context, this fact can be summarized as follows. For two given permutation graphs G_1 and G_2 , let \mathcal{T}_1 and \mathcal{T}_2 be their modular decomposition trees. Then $G_1 \sim G_2$ if and only if (1) \mathcal{T}_1 and \mathcal{T}_2 satisfy $\mathcal{T}_1 \sim \mathcal{T}_2$ (as labeled trees), and (2) corresponding prime modules are isomorphic.

Canonical String Representation. The modular decomposition tree \mathcal{T} for a permutation graph $G = (V, E)$ is the canonical form. As considered for interval graphs, we again introduce a *canonical string representation* for a given permutation graph as follows.

We first consider the case that $G = (V, E)$ is a prime module. In this case, as mentioned, G has a unique line representation up to reversal, and hence G has two representations given by two permutations π and π' with $\pi = \pi'^{-1}$. Each permutation can be represented by a string of length n such that every integer $i \in \{1, \dots, n\}$ appears exactly once. (E.g., P_3 is represented by either 231 or 312.) Therefore, we can choose lexicographically smaller one of π and π' as the canonical string representation of G . (E.g., the canonical string representation of P_3 is 231.)

Now we turn to the general case. This case is similar to the case of interval graphs. We first fix the drawing of the modular decomposition tree according to a total ordering. Then, we can fix the ordering of modules, and then the corresponding line representation is uniquely determined. We then relabel all vertices in V such that they appear as $1, 2, \dots, n$ on L_1 . From this line representation, we can obtain the unique permutation π on L_2 . We regard this π as the canonical string representation of G .

Now the following theorem is straightforward from the results in [4–6] and definitions above.

Theorem 6. *Let $G = (V, E)$ be any permutation graph with $|V| = n$ and $|E| = m$. (1) the canonical modular decomposition tree and the canonical string representation of G can be computed in $O(n + m)$ time. (2) Two permutation graphs G_1 and G_2 are isomorphic if and only if $\pi_1 = \pi_2$, where π_i is the canonical string representation of G_i .*

5.2 Parent-Child Relationship

As shown in Lemma 2, for any given permutation graph $G = (V, E)$ with $G \not\sim K_n$, there is at least one edge $e = \{u, v\}$ with $e \notin E$ such that $G + e$ is also a permutation graph. Therefore, we can use the same idea used in interval graphs. For a given permutation graph G , let \mathcal{T} be the canonical modular decomposition tree of G . We assume that we relabel G according to its canonical string representation, and \mathcal{T} is the corresponding tree. It can be obtained from the original permutation graph in linear time by Theorem 6. Now we let $\hat{E} = \{e = \{u, v\} \mid G + e \text{ is a permutation graph}\}$. Let \hat{e} be the lexicographically smallest element in \hat{E} . We define the unique parent of G by $G + \hat{e}$.

Theorem 7. *Let $G = (V, E)$ be any permutation graph with $|V| = n$ and $|E| = m$ except K_n . Then its parent can be computed in $O(n^2(n + m))$ time.*

5.3 Algorithm Analysis

We here turn to analyze the algorithm. Replacing Theorem 4 by Theorem 7, the analysis is as the same as the case on interval graphs. Therefore, we obtain the following theorem and corollary.

Theorem 8. *We can enumerate every nonisomorphic permutation graph of n vertices. Each permutation graph is enumerated in $O(n^6)$ time delay.*

Corollary 2. *The algorithm in Theorem 8 can be modified to enumerate (1) connected graphs, and/or (2) at most n vertices. In any variant, the delay is not changed from $O(n'^6)$, where n' is the number of vertices of the output graph.*

6 Experimental Results

We implemented the proposed algorithms. The number of vertices and the number of non-isomorphic graphs are summarized as follows, and all these graphs are available at <http://www.jaist.ac.jp/~uehara/graphs>.

# of vertices	1	2	3	4	5	6	7	8	9	10	11	12
# of interval graphs	1	2	4	10	27	92	369	1807	10344	67659	491347	3894446
# of connected int. graphs	1	1	2	5	15	56	250	1328	8069	54962	410330	3317302
# of permutation graphs	1	2	4	11	33	138	-	-	-	-	-	-
# of conn. perm. graphs	1	1	2	6	20	101	-	-	-	-	-	-

7 Concluding Remarks

We propose a general framework that enumerates all nonisomorphic elements in a graph class in which graph isomorphism can be solved in polynomial time. As applications, we give two implementations of the framework for interval graphs and permutation graphs. The first open problem is efficiency. The implementations for the graph classes ran in $O(n^6)$ time, and the real implementation ran up to some certain n , and we succeeded to give real catalogs for these classes. If we can improve running time, we can list up to larger n . The other future work is to extend this framework to more general classes. Even if graph isomorphism cannot be solved in polynomial time, we may enumerate all nonisomorphic graphs up to some certain n for some simple graph classes.

Acknowledgements. This work was supported by JSPS KAKENHI Grant Numbers 26330009, 24106004, 16K1606, and 17H06287, and JST CREST JPMJCR1402.

References

1. Avis, D., Fukuda, K.: Reverse search for enumeration. *Discrete Appl. Math.* **65**, 21–46 (1996)
2. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using PQ -tree algorithms. *J. Comput. Syst. Sci.* **13**, 335–379 (1976)
3. Brandstädt, A., Le, V.B., Spinrad, J.P.: *Graph Classes: A Survey*. SIAM, Philadelphia (1999)
4. Colbourn, C.J.: On testing isomorphism of permutation graphs. *Networks* **11**, 13–21 (1981)
5. Crespelle, C., Paul, C.: Fully dynamic algorithm for recognition and modular decomposition of permutation graphs. *Algorithmica* **58**(2), 405–432 (2009)
6. Gallai, T.: Transitiv orientierbare Graphen. *Acta Mathematica Academiae Scientiarum Hungaricae* **18**, 25–66 (1967)
7. Golumbic, M.C.: *Algorithmic Graph Theory and Perfect Graphs*. *Annals of Discrete Mathematics*, vol. 57, 2nd edn. Elsevier, Amsterdam (2004)
8. Hanlon, P.: Counting interval graphs. *Trans. Am. Math. Soc.* **272**(2), 383–426 (1982)
9. Hegernes, P.: Personal communication (2013)
10. Nakano, S., Uno, T.: Constant time generation of trees with specified diameter. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) *WG 2004*. LNCS, vol. 3353, pp. 33–45. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30559-0_3
11. Kiyomi, M., Kijima, S., Uno, T.: Listing chordal graphs and interval graphs. In: Fomin, F.V. (ed.) *WG 2006*. LNCS, vol. 4271, pp. 68–77. Springer, Heidelberg (2006). https://doi.org/10.1007/11917496_7
12. Köbler, J., Schöning, U., Torán, J.: *The Graph Isomorphism Problem: Its Structural Complexity*. Birkhäuser, Basel (1993)
13. Korte, N., Möhring, R.H.: An incremental linear-time algorithm for recognizing interval graphs. *SIAM J. Comput.* **18**(1), 68–81 (1989)
14. Lueker, G.S., Booth, K.S.: A linear time algorithm for deciding interval graph isomorphism. *J. ACM* **26**(2), 183–195 (1979)
15. McConnell, R.M., Spinrad, J.P.: Modular decomposition and transitive orientation. *Discrete Math.* **201**, 189–241 (1999)
16. Saitoh, T., Otachi, Y., Yamanaka, K., Uehara, R.: Random generation and enumeration of bipartite permutation graphs. *J. Discrete Algorithms* **10**, 84–97 (2012). <https://doi.org/10.1016/j.jda.2011.11.001>
17. Saitoh, T., Yamanaka, K., Kiyomi, M., Uehara, R.: Random generation and enumeration of proper interval graphs. *IEICE Trans. Inf. Syst.* **E93–D**(7), 1816–1823 (2010)
18. Spinrad, J.P.: *Efficient Graph Representations*. American Mathematical Society, Providence (2003)
19. Uehara, R., Toda, S., Nagoya, T.: Graph isomorphism completeness for chordal bipartite graphs and strongly chordal graphs. *Discrete Appl. Math.* **145**(3), 479–482 (2004)

Secret Key Amplification from Uniformly Leaked Key Exchange Complete Graph

Tatsuya Sasaki¹(✉), Bateh Mathias Agbor¹, Shingo Masuda¹,
Yu-ichi Hayashi², Takaaki Mizuki³, and Hideaki Sone³

¹ Graduate School of Information Sciences, Tohoku University,
6-3-09 Aramaki-Aza-Aoba, Aoba, Sendai 980-8579, Japan

`tatsuya.sasaki.p2@dc.tohoku.ac.jp`

² Nara Institute of Science and Technology,
8916-5 Takayama, Ikoma, Nara 630-0192, Japan

³ Cyberscience Center, Tohoku University,
6-3 Aramaki-Aza-Aoba, Aoba, Sendai 980-8578, Japan
`tm-paper+plkzen@g-mail.tohoku-university.jp`

Abstract. We assume that every pair of n players has shared a one-bit key in advance, and that each key has been completely exposed to an eavesdropper, Eve, independently with a fixed probability p (and, thus, is perfectly secure with a probability of $1 - p$). Using these pre-shared, possibly leaked keys, we want two designated players to share a common one-bit secret key in cooperation with other players so that Eve's knowledge about the generated secret key will be as small as possible. The existing protocol, called the st-flow protocol, achieves this, but the specific probability that Eve knows the generated secret key is unknown. In this study, we answer this problem by showing the exact leak probability as a polynomial in p for any number n of players.

Keywords: Key exchange graph · st-numbering
Key agreement protocol · Privacy amplification
Network reliability problem

1 Introduction

Assume that there are n players and an eavesdropper, Eve, where several pairs of players have shared one-bit secret keys in advance. This situation is represented by an undirected multigraph $G = (V, E)$, such that each player corresponds to a vertex $v \in V$, and each pair of players sharing a key corresponds to an edge $e \in E$. The graph G is called a *key exchange graph*, and we express the key corresponding to an edge e as $k_e \in \{0, 1\}$. Figure 1 shows an example of a key exchange graph, G^{ex} , in which players s and t , s and v , and v and t have shared one-bit keys k_{st} , k_{sv} , and k_{vt} , respectively.

The pre-shared keys in the key exchange graph $G = (V, E)$ were obtained using, for example, the Diffie-Hellman key exchange, the RSA algorithm, quantum cryptography, snail mail, e-mail, or face-to-face communication. Furthermore, we assume that some of these keys have been leaked to Eve according to

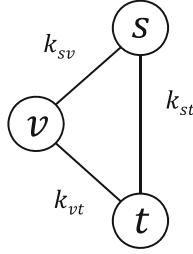


Fig. 1. A key exchange graph $G^{\text{ex}} = (\{s, t, v\}, \{st, sv, vt\})$

a certain probability distribution (cf. [12, 13]). That is, a *leaked-edge set* $F \subseteq E$ occurs according to a *leak distribution* \mathcal{L} , meaning that all keys corresponding to edges in F are known to Eve (while all keys corresponding to edges in $E - F$ are not known to Eve)¹. We call the pair (G, \mathcal{L}) of the key exchange graph $G = (V, E)$ and the leak distribution \mathcal{L} on $2^{|E|}$ a *partially leaked key exchange graph*.

As an introduction to partially leaked key exchange graphs, we first define some leak distributions for the graph G^{ex} depicted in Fig. 1 by giving specific probability distributions on $2^{\{st, sv, vt\}}$. The simplest is perhaps the uniform distribution, where each key leaks independently with a probability of a fixed value, say 0.1; in this case, the leaked-edge set $\{k_{st}, k_{sv}, k_{vt}\}$ occurs with a probability of $0.1 \times 0.1 \times 0.1$, $\{k_{st}, k_{sv}\}$ occurs with a probability of $0.1 \times 0.1 \times (1 - 0.1)$, and so on. The leak distribution is then \mathcal{L}_1 , as shown in Table 1.

Table 1. Leak distribution \mathcal{L}_1

Leaked-edge set	$\{k_{st}, k_{sv}, k_{vt}\}$	$\{k_{st}, k_{sv}\}$	$\{k_{st}, k_{vt}\}$	$\{k_{sv}, k_{vt}\}$
Occurrence probability	0.001	0.009	0.009	0.009
	$\{k_{st}\}$	$\{k_{sv}\}$	$\{k_{vt}\}$	\emptyset
	0.081	0.081	0.081	0.729

In general, pre-shared keys might not leak independently. For example, for graph G^{ex} , the leak distribution \mathcal{L}_2 shown in Table 2 is also conceivable. Here, the probability that all keys leak is 0.1, the probability that only k_{st} leaks is 0.2, and the probability that no keys leak is 0.7 (other events never occur).

We have shown two examples $(G^{\text{ex}}, \mathcal{L}_1)$ and $(G^{\text{ex}}, \mathcal{L}_2)$ of partially leaked key exchange graphs. The problem we consider in this study is as follows. Given a

¹ In this paper, the expression “Eve does not know key k ” means the key is completely unknown to Eve; that is, she cannot determine whether $k = 0$ or $k = 1$ with a probability of more than $1/2$.

Table 2. Leak distribution \mathcal{L}_2

Leaked-edge set	$\{k_{st}, k_{sv}, k_{vt}\}$	$\{k_{st}\}$	\emptyset
Occurrence probability	0.1	0.2	0.7

partially leaked key exchange graph (G, \mathcal{L}) , along with two players s and t in G , we want s and t to share a common one-bit secret key $u \in \{0, 1\}$ in cooperation with other players so that the probability that Eve knows the generated secret key u is small. When a protocol \mathcal{P} results in s and t sharing a key u for (G, \mathcal{L}) , we denote by $\mathcal{E}_{\mathcal{P}}(G, \mathcal{L}, s, t)$, or simply $\mathcal{E}_{\mathcal{P}}(G, \mathcal{L})$, the probability that Eve knows u shared by the two players². Here, we do not care if players other than s and t happen to know the generated secret key u . We assume that all players and Eve have an authenticated public channel, as is usually assumed in privacy amplification schemes (e.g., [2, 4, 16]).

Consider the partially leaked key exchange graph $(G^{\text{ex}}, \mathcal{L}_1)$ as an example. If we use the pre-shared key k_{st} as the secret key u , then the leak probability of u to Eve is 0.1 (because each key in the graph leaks independently with a probability of 0.1). Denoting this trivial protocol by \mathcal{P}_1 , we have

$$\mathcal{E}_{\mathcal{P}_1}(G^{\text{ex}}, \mathcal{L}_1) = 0.1.$$

As another example, consider a protocol \mathcal{P}_2 that uses two keys k_{sv} and k_{vt} . That is, player s first selects a random bit u , and then sends it to player v by announcing $u \oplus k_{sv}$ publicly, that is, using k_{sv} as a one-time pad [14]. Next, player v sends the received bit u to player t in a similar manner, resulting in s and t sharing the same one-bit secret key u . In this case, the leak probability to Eve is

$$\mathcal{E}_{\mathcal{P}_2}(G^{\text{ex}}, \mathcal{L}_1) = 1 - (1 - 0.1)^2 = 0.19,$$

because the generated secret key u known to Eve only when at least one of keys k_{sv} and k_{vt} leaks. (Note that player v learns the secret key u here, but this is not relevant, as mentioned above.)

Combining the two protocols \mathcal{P}_1 and \mathcal{P}_2 , we immediately have protocol \mathcal{P}_3 that uses all three pre-shared keys, as follows. We use protocol \mathcal{P}_1 to share secret key u_1 , and use \mathcal{P}_2 to share u_2 between s and t . Then, s and t can obtain the desired secret key $u = u_1 \oplus u_2$ by XORing. Then, the probability that Eve knows u is calculated as

$$\mathcal{E}_{\mathcal{P}_3}(G^{\text{ex}}, \mathcal{L}_1) = 0.1 \times 0.19 = 0.019.$$

Figure 2 illustrates the three protocols \mathcal{P}_1 , \mathcal{P}_2 , and \mathcal{P}_3 using directed edges. Of these three protocols, \mathcal{P}_3 is the best in terms of the leak probability of u to Eve. In fact, there exists no protocol \mathcal{P} such that

$$\mathcal{E}_{\mathcal{P}}(G^{\text{ex}}, \mathcal{L}_1) < \mathcal{E}_{\mathcal{P}_3}(G^{\text{ex}}, \mathcal{L}_1) = 0.019.$$

² Note that the notation \mathcal{P} in this paper denotes a protocol, not a power set.

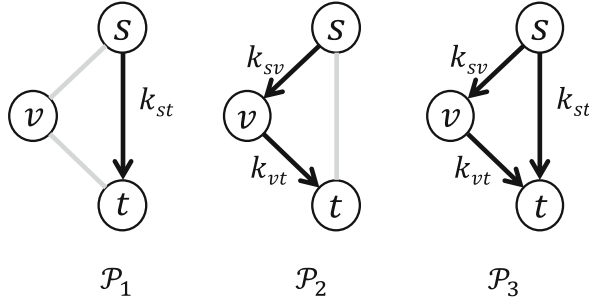


Fig. 2. Illustrative protocols \mathcal{P}_1 , \mathcal{P}_2 , and \mathcal{P}_3

Here, \mathcal{P}_3 is simply the existing st-flow protocol [10] for the case of the partially leaked key exchange graph $(G^{\text{ex}}, \mathcal{L}_1)$. The st-flow protocol always uses all pre-shared keys in a given partially leaked key exchange graph, and it has been proven that the leak probability to Eve through the st-flow protocol is always minimized [10]. Therefore, for any partially leaked key exchange graph (G, \mathcal{L}) , there is no protocol \mathcal{P} such that $\mathcal{E}_{\mathcal{P}}(G, \mathcal{L}) < \mathcal{E}_{\text{st-flow}}(G, \mathcal{L})$ (where the subscript “st-flow” denotes the st-flow protocol). See Sect. 2 for further detail.

Although the st-flow protocol is optimal in terms of the leak probability to Eve, it does have two issues.

1. The st-flow protocol exhausts all pre-shared keys in a given key exchange graph. Therefore, it cannot deal with the case in which “designated players do not need to minimize the leak probability so that they can leave some keys for future use.” For example, people might be satisfied with protocol \mathcal{P}_1 , which has a leak probability of 0.1, leaving two keys fresh, rather than achieving a leak probability of 0.019 using protocol \mathcal{P}_3 , which exhausts all three keys. Moreover, as described above, using two keys does not always reduce the leak probability to Eve below that of using one key. Thus, how to select pre-shared keys is an important issue.
2. There is no existing method that can determine the value $\mathcal{E}_{\text{st-flow}}(G, \mathcal{L})$ efficiently, that is, the probability that Eve knows u via the st-flow protocol, for a given partially leaked key exchange graph (G, \mathcal{L}) . (We know that it is minimized, but we do not know what the value is.)

One of the main difficulties in solving the above problems for any general partially leaked key exchange graph (G, \mathcal{L}) is that describing a leak distribution \mathcal{L} could need an exponential size in the number of edges in G . Therefore, we restrict our attention to the class of *uniformly leaked key exchange complete graphs*. That is, for a partially leaked key exchange graph (G, \mathcal{L}) , we assume that G is a complete graph K_n (with $n = |G|$), meaning that every pair of players has a pre-shared key, and \mathcal{L} is uniform, such that each key leaks independently with a fixed probability p . We denote such a uniformly leaked key exchange complete graph as (K_n, p) . Thus, the previous example of the leaked key exchange graph $(G^{\text{ex}}, \mathcal{L}_1)$ can be written as $(K_3, 0.1)$.

In this study, we aim to solve the two above-mentioned problems for uniformly leaked key exchange complete graphs (K_n, p) . Specifically, in Sect. 3, we give a straightforward protocol, denoted by $\mathcal{P}_{\text{path}}^\ell$, to provide a simple way of selecting ℓ pre-shared keys, with $\ell \leq 2n - 3$. As shown later, the leak probability is calculated as

$$\mathcal{E}_{\mathcal{P}_{\text{path}}^\ell}(K_n, p) = p(2p - p^2)^{\frac{\ell-1}{2}},$$

for an odd number ℓ .

Now, how “relatively” small is $\mathcal{E}_{\mathcal{P}_{\text{path}}^\ell}(K_n, p)$? To see this, we compare it with $\mathcal{E}_{\text{st-flow}}(K_n, p)$, which is the minimum leak probability. The simple protocol $\mathcal{P}_{\text{path}}^\ell$ uses at most $2n - 3$ keys, while the st-flow protocol uses all $(n^2 - n)/2$ keys in a complete graph. This relates to the second problem above. In Sect. 4, we give a complete answer to the second problem by proposing a method to produce polynomials such as

$$\begin{aligned} \mathcal{E}_{\text{st-flow}}(K_{10}, p) = & 2p^9 + 16p^{16} - 17p^{17} + 56p^{21} - 184p^{23} + 240p^{24} \\ & + 70p^{25} - 504p^{27} - 392p^{28} + 812p^{29} - 840p^{30} \\ & - 1736p^{31} + 2464p^{32} + 6314p^{33} - 11424p^{34} + 24304p^{35} \\ & - 10640p^{36} - 36260p^{37} - 43680p^{39} + 263760p^{40} \\ & - 172200p^{41} - 272160p^{42} + 433440p^{43} - 221760p^{44} \\ & + 40320p^{45}, \end{aligned}$$

where $n = 10$. Of course, we can efficiently produce polynomials for any number of players n using this method. Our method utilizes a well-known solution to the two-terminal network reliability problem [3].

We compare the leak probabilities $\mathcal{E}_{\mathcal{P}_{\text{path}}^\ell}(K_n, p)$ and $\mathcal{E}_{\text{st-flow}}(K_n, p)$ in Sect. 5 to describe the trade-off between the number of keys used and the leak probability. Then, considering this trade-off, we analyze typical cases, and demonstrate what kind of key selection would be effective.

Before presenting our results in Sects. 3, 4 and 5, we describe the properties of the st-flow protocol [10] in Sect. 2.

As described thus far, this study deals with the key-generating problem using a partially leaked key exchange graph, which can be considered a kind of “privacy amplification” from partially leaked key exchange graphs. We assume that each player knows only the pre-shared keys specified by the key exchange graph, and that all communications among players are broadcast and, hence, are overheard by Eve. Several previous studies follow the same assumption (e.g., [8, 11]). Other related studies have achieved secret transmission using partial communication paths, such as Secure Message Transmission or Private Message Transmission, rather than assuming a key exchange graph (e.g., [1, 5–7, 15]). Note that our setting of this paper is different from those: in this paper, all messages are broadcast even to Eve and, hence, our security is derived from pre-shared keys.

2 Known Results

In this section, we explain the properties of the st-flow protocol [10].

Although we omit the details, the st-flow protocol generates a directed graph based on “st-numbering” [9], which minimizes the leak probability of the generated key u to Eve by utilizing all the pre-shared keys. Hence, given an arbitrary partially leaked key exchange graph (G, \mathcal{L}) , it holds that

$$\mathcal{E}_{\mathcal{P}}(G, \mathcal{L}) \geq \mathcal{E}_{\text{st-flow}}(G, \mathcal{L}),$$

for any protocol \mathcal{P} .

It is also known that we can characterize whether the generated key u leaks using a graph theoretic property of leaked-edge sets. Specifically, as shown in Fig. 3(a), if a leaked-edge set F separates players s and t , then the key u leaks to Eve. On the other hand, as shown in Fig. 3(b), if a leaked-edge set F does not separate s and t , then the key u does not leak to Eve.

Therefore, we have the following theorem, where we define the set of all leaked-edge sets that separate s and t as

$$\text{Sep}(s, t; G) = \{F \subseteq E \mid F \text{ separates } s \text{ and } t\}$$

for a graph $G = (V, E)$ and two vertices s and t in G .

Theorem 1 [10]. *Let (G, \mathcal{L}) be a partially leaked key exchange graph. For any protocol \mathcal{P} , it holds that*

$$\mathcal{E}_{\mathcal{P}}(G, \mathcal{L}) \geq \mathcal{E}_{\text{st-flow}}(G, \mathcal{L}) = \sum_{F \in \text{Sep}(s, t; G)} \Pr(F),$$

where $\Pr(F)$ is the probability that leaked-edge set F occurs according to \mathcal{L} .

Although Theorem 1 guarantees that $\mathcal{E}_{\text{st-flow}}(G, \mathcal{L})$ is minimized, how to determine its value analytically is unknown. (We solve this problem in Sect. 4.)

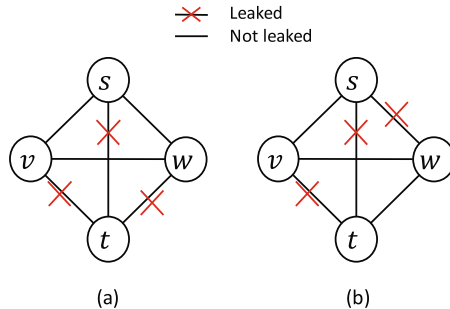


Fig. 3. (a) A leaked-edge set F separates s and t ; (b) F does not separate s and t

3 Simple Protocol for up to $2n - 3$ Keys

As mentioned earlier, the st-flow protocol exhausts all $(n^2 - n)/2$ pre-shared keys in a complete graph. In this section, we demonstrate a simple protocol $\mathcal{P}_{\text{path}}^\ell$ that uses only ℓ pre-shared keys, where $\ell \leq 2n - 3$, for a uniformly leaked key exchange complete graph (K_n, p) , as follows:

1. Select edge st .
2. While the number of selected edges does not exceed ℓ , select two edges that make up a path of length 2 from s to t (as shown in Fig. 4). Note that there are exactly $n - 2$ disjoint paths of length 2 between s and t .
3. If ℓ is an even number of 6 or more, select one edge connecting the vertices, excluding s and t (as shown in Fig. 5).

The leak probability through the path of length 1 is p , and the path of length 2 is $1 - (1 - p)^2 = 2p - p^2$. Hence, the leak probability using protocol $\mathcal{P}_{\text{path}}^\ell$ can be calculated as in the following lemma.

Lemma 1. *Let (K_n, p) be a uniformly leaked key exchange complete graph, and let ℓ be such that $\ell \leq 2n - 3$. If ℓ is an odd number,*

$$\mathcal{E}_{\mathcal{P}_{\text{path}}^\ell}(K_n, p) = p(2p - p^2)^{\frac{\ell-1}{2}}.$$

If ℓ is an even number of 6 or more,

$$\mathcal{E}_{\mathcal{P}_{\text{path}}^\ell}(K_n, p) = p(2p^5 - 5p^4 + 2p^3 + 2p^2)(2p - p^2)^{\frac{\ell-6}{2}}.$$

Furthermore, $\mathcal{E}_{\mathcal{P}_{\text{path}}^2}(K_n, p) = p$, and $\mathcal{E}_{\mathcal{P}_{\text{path}}^4}(K_n, p) = p(2p - p^2)$.

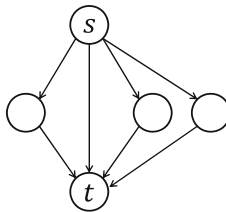


Fig. 4. The selection of keys up to Step 2 of $\mathcal{P}_{\text{path}}^\ell$

The simple protocol $\mathcal{P}_{\text{path}}^\ell$ uses at most $2n - 3$ keys, and while the st-flow protocol uses all $(n^2 - n)/2$ keys. To compare $\mathcal{P}_{\text{path}}^\ell(K_n, p)$ with $\mathcal{E}_{\text{st-flow}}(K_n, p)$, we give a method to compute $\mathcal{E}_{\text{st-flow}}(K_n, p)$ in the next section.

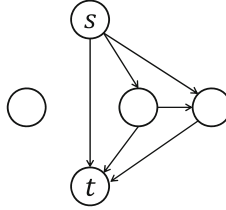


Fig. 5. The selection of keys up to Step 3 of $\mathcal{P}_{\text{path}}^\ell$

4 Finding the Minimum Leak Probability

As mentioned before, there is no known efficient method of obtaining the leak probability to Eve using the st-flow protocol, even for a uniformly leaked key exchange complete graph. In this section, we first show that calculating $\mathcal{E}_{\text{st-flow}}(K_n, p)$ can be reduced to the *two-terminal network reliability problem* [3]. Then, using an existing algorithm to compute the two-terminal network reliability in complete graphs, we provide a recurrence formula that enables us to derive $\mathcal{E}_{\text{st-flow}}(K_n, p)$ for any uniformly leaked key exchange complete graph (K_n, p) .

4.1 Formula for $\mathcal{E}_{\text{st-flow}}(K_n, p)$

To explain (a simplified version of) the two-terminal network reliability problem, we first adopt the following computer network model. A computer network is represented by an undirected multigraph $G = (V, E)$, such that each computer corresponds to a vertex $v \in V$, and each communications link corresponds to an edge $e \in E$. Here, we assume that G is a complete graph K_n , and that each edge fails independently with a fixed probability p . Hence, this situation can be represented by a *probabilistic graph* (K_n, p) . For a probabilistic graph (G, p) and two designated terminals s and t , the *two-terminal network reliability* $R(G, p)$ is defined as the probability that s and t are connected. We obtain the two-terminal network reliability by considering the probability of all “failed-edge” sets, each of which connect the two terminals.

Considering a uniformly leaked key exchange complete graph (K_n, p) as a probabilistic graph (K_n, p) , we have

$$R(K_n, p) + \mathcal{E}_{\text{st-flow}}(K_n, p) = 1. \quad (1)$$

Therefore, the problem of finding $\mathcal{E}_{\text{st-flow}}(K_n, p)$ is equivalent to the two-terminal network reliability problem, which has been well studied in the literature (e.g., [3]).

The known result [3] tells us that

$$R(K_n, p) = 1 - \sum_{j=1}^{n-1} \binom{n-2}{j-1} p^{j(n-j)} A(K_j, p), \quad (2)$$

where $A(G, p)$ is the “all-terminal” reliability, which is obtained recursively by

$$A(K_n, p) = 1 - \sum_{j=1}^{n-1} \binom{n-1}{j-1} p^{j(n-j)} A(K_j, p). \quad (3)$$

Therefore, from Eqs. (1), (2), and (3), we have

$$\begin{aligned} \mathcal{E}_{\text{st-flow}}(K_n, p) &= 1 - R(K_n, p) \\ &= \sum_{j=1}^{n-1} \binom{n-2}{j-1} p^{j(n-j)} A(K_j, p). \end{aligned} \quad (4)$$

4.2 Examples of Polynomials

This section provides examples of polynomials that specify $\mathcal{E}_{\text{st-flow}}(K_n, p)$ derived from the recurrence formula (4) in the previous subsection:

$$\begin{aligned} \mathcal{E}_{\text{st-flow}}(K_4, p) &= 2p^3 + 2p^4 - 5p^5 + 2p^6; \\ \mathcal{E}_{\text{st-flow}}(K_5, p) &= 2p^4 + 6p^6 - 7p^7 - 12p^8 + 18p^9 - 6p^{10}; \\ \mathcal{E}_{\text{st-flow}}(K_6, p) &= 2p^5 + 8p^8 - 3p^9 - 44p^{11} + 20p^{12} + 78p^{13} - 84p^{14} + 24p^{15}; \\ \mathcal{E}_{\text{st-flow}}(K_7, p) &= 2p^6 + 10p^{10} - 11p^{11} + 20p^{12} - 70p^{14} - 80p^{16} \\ &\quad + 340p^{17} - 570p^{19} + 480p^{20} - 120p^{21}; \\ \mathcal{E}_{\text{st-flow}}(K_8, p) &= 2p^7 + 12p^{12} - 13p^{13} + 30p^{15} + 20p^{16} - 102p^{17} + 72p^{18} \\ &\quad - 190p^{19} - 150p^{20} + 420p^{21} + 110p^{22} + 1380p^{23} - 2700p^{24} \\ &\quad - 1050p^{25} + 4680p^{26} - 3240p^{27} + 720p^{28}; \\ \mathcal{E}_{\text{st-flow}}(K_9, p) &= 2p^8 + 14p^{14} - 15p^{15} + 42p^{18} - 70p^{20} + 98p^{21} \\ &\quad - 322p^{23} - 462p^{24} + 1050p^{25} - 1456p^{26} + 1680p^{27} \\ &\quad + 2940p^{28} - 2030p^{29} + 420p^{30} - 19530p^{31} + 21840p^{32} \\ &\quad + 18480p^{33} - 42840p^{34} + 25200p^{35} - 5040p^{36}; \\ \mathcal{E}_{\text{st-flow}}(K_{10}, p) &= 2p^9 + 16p^{16} - 17p^{17} + 56p^{21} - 184p^{23} + 240p^{24} \\ &\quad + 70p^{25} - 504p^{27} - 392p^{28} + 812p^{29} - 840p^{30} \\ &\quad - 1736p^{31} + 2464p^{32} + 6314p^{33} - 11424p^{34} + 24304p^{35} \\ &\quad - 10640p^{36} - 36260p^{37} - 43680p^{39} + 263760p^{40} - 172200p^{41} \\ &\quad - 272160p^{42} + 433440p^{43} - 221760p^{44} + 40320p^{45}. \end{aligned}$$

5 Comparison

In this section, we illustrate a comparison of the st-flow protocol and the simple protocol $\mathcal{P}_{\text{path}}^\ell$ to provide a guide for determining how many pre-shared keys people would use.

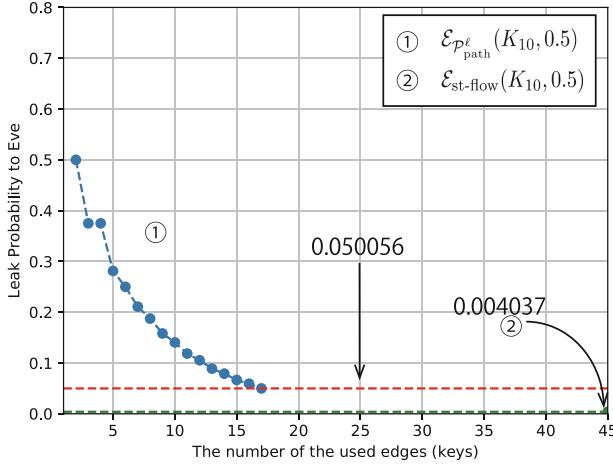


Fig. 6. Leak probabilities when $p = 0.5, n = 10$

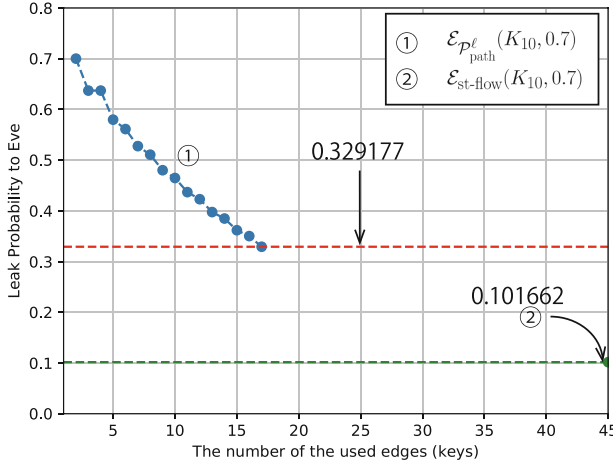


Fig. 7. Leak probabilities when $p = 0.7, n = 10$

The st-flow protocol uses all $(n^2 - n)/2$ pre-shared keys in K_n , while the simple protocol uses at most $2n - 3$ keys. Therefore, the difference is at least $(n^2 - 5n + 6)/2$ keys. Thus, we want to determine whether to use all the keys or leave $(n^2 - 5n + 6)/2$ for future use. The examples presented earlier can be used to calculate the specific leak probability, and people can judge which would be better for their own purpose.

As an example, consider K_{10} , and set the probability p to 0.5. Figure 6 shows the value of $\mathcal{E}_{\text{path}}^\ell(K_{10}, 0.5)$ for every number ℓ , where $\ell \leq 17$, and the value of $\mathcal{E}_{\text{st-flow}}(K_{10}, 0.5)$. Note that the st-flow protocol uses 45 keys.

Now, set the probability p to 0.7. Figure 7 shows the values of $\mathcal{E}_{\mathcal{P}_{path}^\ell}(K_{10}, 0.7)$ and the value of $\mathcal{E}_{\text{st-flow}}(K_{10}, 0.7)$.

We believe that this kind of comparison can help to determine how many keys to use, based on a target leak probability.

6 Conclusion

We first proposed a simple protocol that does not use all available keys. Then, to compare it with the st-flow protocol, we described a method to efficiently compute the value of $\mathcal{E}_{\text{st-flow}}(K_n, p)$ for any uniformly leaked key exchange complete graph (K_n, p) . These results provide a guide for determining how many keys to use.

It would be interesting future work to design methods that select keys effectively for an arbitrary key exchange graph or an arbitrary leak distribution. Furthermore, we assumed in this study that it is not relevant whether other players know the secret key u ; however, it may be desirable not to make such an assumption. For example, using the st-flow protocol or the simple protocol \mathcal{P}_{path}^ℓ , if the selected keys constitute a biconnected graph, u is not known to other players unless collusion occurs.

Acknowledgement. We thank the anonymous referees, whose comments have helped us to improve the presentation of the paper. We thank Mr. Shigehiro Matsuda for his valuable discussions. This work was supported by JSPS KAKENHI Grant Number 15K11983.

References

1. Ahmadi, H., Safavi-Naini, R.: Private message transmission using disjoint paths. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) ACNS 2014. LNCS, vol. 8479, pp. 116–133. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07536-5_8
2. Bennett, C.H., Brassard, G., Crépeau, C., Maurer, U.M.: Generalized privacy amplification. *IEEE Trans. Inf. Theory* **41**(6), 1915–1923 (1995). <https://doi.org/10.1109/18.476316>
3. Colbourn, C.J., Colbourn, C.: *The Combinatorics of Network Reliability*, vol. 200. Oxford University Press, New York (1987)
4. Csiszár, I., Narayan, P.: Secrecy capacities for multiple terminals. *IEEE Trans. Inf. Theory* **50**(12), 3047–3061 (2004). <https://doi.org/10.1109/TIT.2004.838380>
5. Dolev, D., Dwork, C., Waarts, O., Yung, M.: Perfectly secure message transmission. *J. ACM* **40**(1), 17–47 (1993). <https://doi.org/10.1145/138027.138036>
6. Franklin, M.K., Wright, R.N.: Secure communication in minimal connectivity models. *J. Cryptol.* **13**(1), 9–30 (2000). <https://doi.org/10.1007/s001459910002>
7. Franklin, M.K., Yung, M.: Secure hypergraphs: Privacy from partial broadcast. *SIAM J. Discrete Math.* **18**(3), 437–450 (2004). <https://doi.org/10.1137/S0895480198335215>
8. Indo, Y., Mizuki, T., Nishizeki, T.: Absolutely secure message transmission using a key sharing graph. *Discrete Math. Alg. Appl.* **4**(4) (2012). <https://doi.org/10.1142/S179383091250053X>

9. Lempel, A., Even, S., Cederbaum, I.: An algorithm for planarity testing of graphs. In: *Theory of Graphs: International Symposium*, pp. 215–232 (1967)
10. Mizuki, T., Nakayama, S., Sone, H.: An application of st-numbering to secret key agreement. *Int. J. Found. Comput. Sci.* **22**(5), 1211–1227 (2011). <https://doi.org/10.1142/S0129054111008659>
11. Mizuki, T., Sato, T., Sone, H.: A one-round secure message broadcasting protocol through a key sharing tree. *Inf. Process. Lett.* **109**(15), 842–845 (2009). <https://doi.org/10.1016/j.ipl.2009.04.004>
12. Nagaraja, S.: Privacy amplification with social networks. In: Christianson, B., Crispo, B., Malcolm, J.A., Roe, M. (eds.) *Security Protocols 2007*. LNCS, vol. 5964, pp. 58–73. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17773-6_7
13. Ošt’ádal, R., Švenda, P., Matyáš, V.: A new approach to secrecy amplification in partially compromised networks (invited paper). In: Chakraborty, R.S., Matyas, V., Schaumont, P. (eds.) *SPACE 2014*. LNCS, vol. 8804, pp. 92–109. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12060-7_7
14. Vernam, G.S.: Cipher printing telegraph systems for secret wire and radio telegraphic communications. *Trans. Am. Inst. Electr. Eng.* **XLV**, 295–301 (1926)
15. Wang, Y., Desmedt, Y.: Secure communication in multicast channels: The answer to franklin and wright’s question. *J. Cryptol.* **14**(2), 121–135 (2001). <https://doi.org/10.1007/s00145-001-0002-y>
16. Watanabe, S., Matsumoto, R., Uyematsu, T.: Strongly secure privacy amplification cannot be obtained by encoder of slepian-wolf code. *IEICE Trans.* **93**(9), 1650–1659 (2010). http://search.ieice.org/bin/summary.php?id=e93-a_9_1650

Approximating Partially Bounded Degree Deletion on Directed Graphs

Toshihiro Fujito^(✉) , Kei Kimura, and Yuki Mizuno

Toyohashi University of Technology, Toyohashi 441-8580, Japan
{fujito,kimura}@cs.tut.ac.jp, mizuno@algo.cs.tut.ac.jp

Abstract. The BOUNDED DEGREE DELETION problem (BDD) is that of computing a minimum vertex set in a graph $G = (V, E)$ with degree bound $b : V \rightarrow \mathbb{Z}_+$, such that, when it is removed from G , the degree of any remaining vertex v is no larger than $b(v)$. It is a classic problem in graph theory and various results have been obtained including an approximation ratio of $2 + \ln b_{\max}$ [30], where b_{\max} is the maximum degree bound.

This paper considers BDD on *directed* graphs containing *unbounded* vertices, which we call PARTIALLY BOUNDED DEGREE DELETION (PBDD). Despite such a natural generalization of standard BDD, it appears that PBDD has never been studied and no algorithmic results are known, approximation or parameterized. It will be shown that (1) in case all the possible degrees are bounded, in-degrees by b^- and out-degrees by b^+ , BDD on directed graphs can be approximated within $2 + \max_{v \in V} \ln(b^-(v) + b^+(v))$, and (2) although it becomes NP-hard to approximate PBDD better than b_{\max} (even on *undirected* graphs) once unbounded vertices are allowed, it can be within $\max\{2, b_{\max} + 1\}$ when only in-degrees (and none of out-degrees) are partially bounded by b .

Keywords: Approximation algorithms · Bounded Degree Deletion
Partial cover

1 Introduction

The BOUNDED DEGREE DELETION problem is a well-known basic problem in graph theory. It has an application in various areas such as computational biology [15] and property testing [29], whereas its “dual problem” of finding maximum s -plexes, introduced in 1978 [32], has applications in social network analysis [1, 28]. With degree bound of $b \in \mathbb{Z}_+$, b -BOUNDED DEGREE DELETION (or b -BDD for short) is the problem of computing a minimum cost vertex set X in a given weighted graph $G = (V, E)$ such that the degree of any remaining vertex v is bounded by b when all the vertices in X are removed from G .

This work is supported in part by JSPS KAKENHI under Grant Numbers 26330010 and 17K00013.

Clearly, b -BDD is a generalization of the VERTEX COVER (VC) problem, and another generalization of VC has been recently introduced and actively studied. The k -PATH VERTEX COVER (k -PVC) problem [4–6, 22, 24], also known as VERTEX COVER P_k [34–37], P_k -HITTING SET [7], and k -PATH TRANSVERSAL [27], is the problem of computing a minimum vertex set C such that when all the vertices in C are removed from G , there remains no path on k vertices. A subset of vertices in a graph G is called a *dissociation set* if it induces a subgraph with maximum degree at most 1. The maximum cardinality of a dissociation set in G is called the *dissociation number* of G . The problem of computing the dissociation number was introduced by Yannakakis [40], who also proved it to be NP-hard in the class of bipartite graphs. See [31] for a survey on the dissociation number problem. Clearly, VC \equiv 0-BDD \equiv 2-PVC, 1-BDD \equiv 3-PVC (but b -BDD $\not\equiv$ $(b+2)$ -PVC for $b \geq 2$), and a dissociation set is the complement of a 3-PVC (i.e., 1-BDD) solution.

We now summarize below algorithmic results known for b -BDD and related problems.

VC. It is known approximable within $2 - \Theta(1/\sqrt{\log n})$ [23], whereas VC has been shown hard to approximate within $10\sqrt{5} - 21 \approx 1.36$ unless $P = NP$ [13] (or within $2 - \epsilon$ assuming the unique games conjecture [25]).

b -BDD. The first improvement over the simple $(b+2)$ -approximation based on the hitting set formulation was attained in [17] by using the local ratio method and b -BDD was shown approximable within $\max\{2, b+1\}$. Okun and Barak considered more general b -BDD where $b : V \rightarrow \mathbb{Z}_+$ is an arbitrary function, and obtained an approximation bound of $2 + \ln b_{\max}$ by combination of the local ratio method and the greedy multicovering [30], where $b_{\max} = \max_{v \in V} b(v)$. Recently, a new approximation bound of $\max\{2, b_{\max}/2 + 1\}$ was obtained [19].

b -BDD has been extensively studied in parameterized complexity. It has been shown that, when parameterized by the size k of the deletion set, the problem is $W[2]$ -hard for unbounded b and FPT for each fixed $b \geq 0$ [15], whereas, when parameterized by treewidth tw , it is FPT with parameters k and tw , and $W[2]$ -hard with only parameter tw [3]. A linear vertex kernel of b -BDD has been developed by generalizing the Nemhauser-Trotter theorem for VC to b -BDD [10, 15, 39].

Besides, 2-BDD has been recently highlighted under the name of CO-PATH/CYCLE PACKING [9, 10, 16], mostly from the viewpoint of parameterized complexity, due to its important applications in bioinformatics.

3-PVC. It was shown approximable within 2 [36, 37] (or within an expected approximation ratio of 23/11 by a randomized algorithm [24]) in general, and within 1.57 on cubic graphs [35].

1.1 Our Work and Contributions

We generalize BDD in two directions; in one to the problem of *directed* degree bounds, and in the other to the problem where some vertices are allowed to be

of unbounded degree. PARTIALLY b -BOUNDED DEGREE DELETION (b -PBDD) is, given a *directed* graph $G = (V \cup V_0, E)$ and a degree bound $b : V \rightarrow \mathbb{Z}_+$, to compute a minimum cost vertex subset $X \subseteq V \cup V_0$ such that the *in-degree* of any vertex $v \in V$ remaining after all the vertices in X are deleted from G is at most $b(v)$. Notice that the degree bound b is defined only on V and no bound is imposed on V_0 . To the best of our knowledge, neither version, directed BDD nor partial BDD, has been previously studied, in either aspect of approximation complexity or parameterized one, except for the case of 1-BDD on directed graphs, which was shown approximated within 2 [17]. Certainly, 0 -PBDD \equiv 0 -BDD \equiv VC when $V_0 = \emptyset$, but b -PBDD $\not\equiv$ b -BDD even for $b = 1$.

Directed graphs provide more general computational models than undirected graphs, but problems tend to be harder to deal with on the former than the latter. Another type of generalization, in the setting of BDD, is to allow for “don’t care” nodes. In fact the notion of “covering” or “domination” has been generalized to *partial* “covering/domination” and a significant amount of research work has been devoted to such extensions [2, 8, 11, 14, 20, 21, 26, 33], where, instead of complete coverage or domination, only a prescribed fraction of covering or domination is required. Here we consider PBDD having unbounded vertices as defined above to be a natural extension of BDD to the partial version. The current work is partially motivated by the fact that the (logarithmically) bounded performance of the best algorithm for the standard BDD, however, becomes unbounded when applied to the partial version as will be explained in Sect. 4.1.

This paper presents that (1) in case all the possible degrees are bounded, in-degrees by b^- and out-degrees by b^+ (and $V_0 = \emptyset$), BDD on directed graphs can be approximated within $2 + \max_{v \in V} \ln(b^-(v) + b^+(v))$ by generalizing the algorithm of Okun and Barak [30], and (2) although it becomes NP-hard to approximate b -PBDD better than b_{\max} (even on *undirected* graphs) once unbounded degrees are allowed, it can be within $\max\{2, b_{\max} + 1\}$ when only in-degrees (and none of out-degrees) are partially bounded by b .

1.2 Notations and Definitions

For a vertex set X in a digraph $G = (V, E)$, let $E(X) = \{(u, v) \in E \mid \{u, v\} \subseteq X\}$. Let $\delta^-(X)$ denote the set of arcs entering from outside of X to a vertex in X , i.e., $\delta^-(X) = \{(u, v) \in E \mid u \notin X, v \in X\}$ and $\delta(X)$ be the set of arcs incident to a vertex in X , i.e., $\delta(X) = \{(u, v) \in E \mid \{u, v\} \cap X \neq \emptyset\}$. Let $\delta^-(v)$ ($\delta(v)$, resp.) denote $\delta^-(\{v\})$ ($\delta(\{v\})$, resp.). The in-degree and out-degree of v is denoted by $d^-(v)$ ($= |\delta^-(v)|$) and $d^+(v)$, respectively. To restrict arcs under consideration within a certain arc set F , we use $\delta_F^-(X)$ and $d_F^-(v)$ to denote $\delta^-(X) \cap F$ and $|\delta^-(v) \cap F|$, respectively, and $d_{E(X)}^-(v)$ abbreviated to $d_X^-(v)$ for $X \subseteq V$. For the set of neighboring vertices of $u \in V$, let $N^+(u)$ and $N^-(u)$ denote $\{v \in V \mid (u, v) \in E\}$ and $\{v \in V \mid (v, u) \in E\}$, respectively.

We also use shorthand notations for functions b , d^- , and \bar{w} (to be defined in Sect. 3) defined on V and $Z \subseteq V$ such as $b(Z) = \sum_{v \in Z} b(v)$, $d^-(Z) = \sum_{v \in Z} d^-(v)$, and $\bar{w}(Z) = \sum_{v \in Z} \bar{w}(v)$.

2 Approximating PBDD via Submodular Optimization

Assume that $b(v) \leq d^-(v), \forall v \in V$, for the rest of paper as one can always reset $b(v)$ to $d^-(v)$, without loss of generality, if $b(v) > d^-(v)$. A vertex $v \in V$ is called a *tight node* in what follows if $d^-(v) = b(v)$ (and it is *untight* if $d^-(v) > b(v)$).

For a directed graph $G = (V \cup V_0, E)$ and $b : V \rightarrow \mathbb{Z}_+$, define the rank $r : 2^E \rightarrow \mathbb{Z}_+$ of $F \subseteq E$ such that

$$r(F) = \sum_{v \in V} \min\{d_F^-(v), b(v)\} + \sum_{v \in V_0} d_F^-(v).$$

Then (E, r) is a matroid, a direct sum of *partition matroids* and *free matroids*, and an arc set $F \subseteq E$ is independent iff $d_F^-(v) \leq b(v), \forall v \in V$. Thus, PBDD is the problem of computing $X \subseteq V$ of minimum cost such that the arc set induced by $V - X$ is independent in (E, r) .

Definition 1. For a matroid (E, r) let $r^d : 2^E \rightarrow \mathbb{Z}_+$ be such that

$$r^d(S) = |S| - (r(E) - r(E \setminus S)).$$

Then, r^d is a matroid rank function and (E, r^d) is called the dual of (E, r) .

Proposition 1. Let (E, r) be the matroid defined by (G, b) as above.

- $r(E) = b(V) + d^-(V_0)$ (assuming that $b(v) \leq d^-(v), \forall v \in V$).
- In the dual matroid (E, r^d) ,
 - $r^d(F) = |F| - (r(E) - r(E \setminus F))$

$$= \sum_{v \in V} \left(d_F^-(v) - \min\{b(v), d^-(v)\} + \min\{b(v), d_{E \setminus F}^-(v)\} \right)$$

$$+ \sum_{v \in V_0} \left(d_F^-(v) - d^-(v) + d_{E \setminus F}^-(v) \right)$$

$$= \sum_{v \in V} \left(\min\{b(v) + d_F^-(v), d^-(v)\} - \min\{b(v), d^-(v)\} \right)$$

$$= \sum_{v \in V} \min\{d_F^-(v), d^-(v) - b(v)\}.$$
 - $r^d(E) = |E| - r(E) = |E| - b(V) - d^-(V_0) = d^-(V) - b(V).$
 - $r^d(\delta(v)) = \sum_{u \in V} \min\{d_{\delta(v)}^-(u), d^-(u) - b(u)\}$

$$= \begin{cases} d^-(v) - b(v) + (\# \text{ of untight nodes in } N^+(v) \cap V) & \text{if } v \in V \\ (\# \text{ of untight nodes in } N^+(v) \cap V) & \text{if } v \in V_0 \end{cases}$$

Let $X \subseteq V \cup V_0$ be partitioned to \tilde{X}, X_t , and X_0 s.t. $X_0 = X \cap V_0$, $X_t = \{v \in X \setminus X_0 \mid v \text{ is tight}\}$, and $\tilde{X} = X \setminus (X_t \cup X_0)$. Likewise, for $Y = (V \cup V_0) \setminus X$

let $Y = \tilde{Y} \cup Y_t \cup Y_0$ s.t. $Y_0 = Y \cap V_0$, $Y_t = \{v \in Y \setminus Y_0 \mid v \text{ is tight}\}$, and $\tilde{Y} = Y \setminus (Y_t \cup Y_0)$. Then, since

$$\begin{aligned} \sum_{v \in X} r^d(\delta(v)) &= \sum_{v \in \tilde{X}} r^d(\delta(v)) + \sum_{v \in X_t} r^d(\delta(v)) + \sum_{v \in X_0} r^d(\delta(v)) \\ &= \sum_{v \in \tilde{X}} (d^-(v) - b(v)) + \sum_{v \in X} (\# \text{ of untight nodes in } N^+(v) \cap V), \end{aligned}$$

we have

Proposition 2.

$$\sum_{v \in X} r^d(\delta(v)) = d^-(\tilde{X}) - b(\tilde{X}) + \sum_{v \in X} \left| N^+(v) \cap (\tilde{X} \cup \tilde{Y}) \right|.$$

Note: Propositions 1 and 2 will be useful in proof of Lemma 1.

In general a subset $F \subseteq E$ is independent in a matroid iff F is spanning in its dual matroid. Thus, $X \subseteq V$ is a b -PBDD solution in $G = (V \cup V_0, E)$ iff $\delta(X)$ is spanning in (E, r^d) . Therefore, b -PBDD on $G = (V, E)$ can be reduced to the problem of computing $X \subseteq V$ of minimum cost such that $\delta(X)$ is spanning in (E, r^d) . More formally,

Proposition 3. Define $f : 2^V \rightarrow \mathbb{Z}_+$ such that $f(W) = r^d(\delta(W))$. b -PBDD on $G = (V, E)$ can be formulated as the problem of computing $X \subseteq V$ of minimum cost such that $f(X) = f(V)$.

It is known that f as defined above is nondecreasing and *submodular*, and the problem of computing minimum $X \subseteq V$ satisfying $f(X) = f(V)$ for such a function f is known as the *submodular set cover* problem.

Definition 2. Let f be a nondecreasing submodular set function defined on the subsets of a finite ground set N , and w_j be a nonnegative cost associated with each element $j \in N$. The SUBMODULAR SET COVER problem (SSC) is to compute:

$$\min_{S \subseteq N} \left\{ \sum_{j \in S} w_j \mid f(S) = f(N) \right\}.$$

The greedy algorithm, together with its performance analysis, is perhaps the most well-known heuristic for general SSC [38], but the primal-dual algorithm based on the following LP relaxation of SSC and its dual LP is also known to deliver better solutions for some of more specific SSC problems (See [18] for more details).

$$(P) \quad \min \sum_{j \in N} w_j x_j$$

subject to:

$$\sum_{j \in N-S} f_S(j) x_j \geq f_S(N-S), \forall S \subseteq N$$

$$x_j \geq 0, \quad \forall j \in N$$

$$(D) \quad \max \sum_{S \subseteq N} f_S(N-S) y_S$$

subject to:

$$\sum_{S: j \notin S} f_S(j) y_S \leq w_j, \forall j \in N$$

$$y_S \geq 0, \quad \forall S \subseteq N$$

Here and in the algorithm called PD, the *contraction* of f onto $N - S$ is the function f_S defined on 2^{N-S} such that $f_S(X) = f(X \cup S) - f(S)$ for any $S \subseteq N$. If f is nondecreasing and submodular on N , so is f_S on $N - S$, and thus, another submodular set cover instance $(N - S, f_S)$ can be derived for any $S \subseteq N$. The performance of PD for general SSC can be estimated by the following theorem.

Theorem 1 ([18]). *The performance ratio of the primal-dual algorithm PD for an SSC instance (N, f) is bounded by*

$$\max \left\{ \frac{\sum_{j \in X} f_S(j)}{f_S(N - S)} \right\}$$

where \max is taken over any $S \subseteq N$ and any minimal solution X in $(N - S, f_S)$.

It is more convenient, when applying Theorem 1 to an instance (G, b) of b -PBDD, to use it in the following form.

Corollary 1. *The performance ratio of PD, when applied to an instance $(G = (V, E), b)$ of b -PBDD, is bounded by*

$$\max \left\{ \frac{\sum_{v \in X} r^d(\delta(v))}{r^d(E)} \right\}$$

where (E, r) is the matroid defined by an instance (G, b) and \max is taken over any graph G and any minimal solution X in G .

Proof. Consider the graph $\bar{G} = G - S$ obtained from G by removing all the vertices in S , and reformulate b -PBDD on $\bar{G} = (\bar{V}, \bar{E})$, where $\bar{V} = (V \cup V_0) - S$, $\bar{E} = E - \delta(S)$, as an SSC instance (\bar{E}, \bar{f}) . To do so, let $\bar{r} : 2^{\bar{E}} \rightarrow \mathbb{Z}_+$ be the rank function of the matroid defined by (\bar{E}, \bar{b}) , such that $\bar{r}(F) = \sum_{v \in \bar{V}} \min\{\bar{d}_F^-(v), \bar{b}(v)\}$ for $F \subseteq \bar{E}$, \bar{r}^d be the dual of \bar{r} , and $\bar{f}(T) = \bar{r}^d(\bar{\delta}(T))$ for $T \subseteq \bar{V}$ (Note: Here, $\bar{\delta}(T) = \delta_{\bar{E}}(T)$, $\bar{d}(v) = d_{\bar{E}}(v)$, $\bar{b}(v) = \min\{b(v), \bar{d}^-(v)\}$ for all $T \subseteq \bar{V}$ and $v \in \bar{V}$). It can be shown then that $f_S(T) = \bar{f}(T)$ for any $S \subseteq V \cup V_0$ and $T \subseteq (V \cup V_0) - S$, and in particular, $\bar{f}(v) = f_S(v)$, $\forall v \in \bar{V}$, and $\bar{f}(\bar{V}) = f_S((V \cup V_0) - S)$. Hence, we have

$$\max_{S \subseteq V \cup V_0} \left\{ \frac{\sum_{v \in X} f_S(v)}{f_S((V \cup V_0) - S)} \right\} = \max \left\{ \frac{\sum_{v \in X} \bar{f}(v)}{\bar{f}(\bar{V})} \right\} \quad (1)$$

where \max in RHS is taken over any subgraph \bar{G} of G induced by $\bar{V} \subseteq V \cup V_0$ and any minimal b -PBDD solution X in \bar{G} . It thus follows from Theorem 1 and Eq. (1) that the performance ratio of PD, when applied to b -PBDD, can be estimated by bounding

$$\frac{\sum_{v \in X} f(v)}{f(V \cup V_0)} = \frac{\sum_{v \in X} r^d(\delta(v))}{r^d(E)}$$

for any graph $G = (V \cup V_0, E)$ and any minimal solution X in G . \square

3 Fully Bounded Degree Deletion

It can be observed, modifying the undirected instance to be used in Sect. 4.1 to a directed one, that the greedy set cover approximation is embeddable even if all the in-degrees are bounded in directed graphs. On the other hand, BDD on directed graphs where both in-degree and out-degree are bounded at every vertex can be approximated in much the same way as in the case of undirected graphs. To explain this, suppose all the possible degree bounds are imposed on directed graph $G = (V, E)$, that is, the in-degree of v by $b^- : V \rightarrow \mathbb{Z}_+$ and the out-degree of v by $b^+ : V \rightarrow \mathbb{Z}_+$ for all the vertices $v \in V$ (and $V_0 = \emptyset$ here). Construct G_D from G by replacing each vertex v by two, v_1 and v_2 , connecting all the incoming arcs of v to v_1 while outgoing arcs to v_2 . When arc orientations are ignored, G_D becomes an undirected bipartite graph. An approximate solution for G can be computed by applying an existing algorithm \mathcal{A} to G_D , and taking v into a solution iff either v_1 or v_2 (or both) in G_D is chosen by \mathcal{A} . This way of reducing directed BDD to undirected one yields a 2ρ -approximation when \mathcal{A} is a ρ -approximation because the optimum with respect to G_D is bounded by twice the optimum with respect to G . So, the fully bounded version of BDD is approximable within $4 + 2 \ln \max_{v \in V} \{b^-(v), b^+(v)\}$ by running the Okun-Barak algorithm as \mathcal{A} .

The reduction based approach above can be further refined by rebuilding the Okun-Barak approach within the current framework of submodular optimization. Consider the partition matroids (E, r_-) and (E, r_+) , defined both on E , based on b^- and b^+ , respectively. Here, $X \subseteq V$ is a solution iff $\delta(X)$ is spanning in both (E, r_-) and (E, r_+) , where r_-^d and r_+^d are the dual rank functions of r_- and r_+ , respectively. Define $f : 2^V \rightarrow \mathbb{Z}_+$ such that $f(X) = r_-^d(\delta(X)) + r_+^d(\delta(X))$. Then, f is nondecreasing and submodular, and $X \subseteq V$ is a solution iff $f(X) = r_-^d(\delta(X)) + r_+^d(\delta(X)) = r_-^d(E) + r_+^d(E) = f(V)$. Therefore, the problem can be reduced to SSC (V, f, w) .

Let us adopt the following strategy of two stage approximation from [30]; first apply the local ratio method and then the greedy method for SSC.

1st stage. Suppose $d^-(v) > b^-(v)$ for some $v \in V$. Let $S^- = \{v\} \cup N^-(v)$ and consider the subgraph $G[S^-]$ of G induced by S^- . Since any solution for G including an optimal one must contain v or otherwise, at least $(d^-(v) - b^-(v))$ from $N^-(v)$, we have a valid inequality

$$(d^-(v) - b^-(v))x_v + \sum_{u \in N^-(v)} x_u \geq (d^-(v) - b^-(v))$$

for any v with $d^-(v) > b^-(v)$, where $\mathbf{x} \in \{0, 1\}^V$ denotes an incidence vector of a vertex subset. We may thus apply the local ratio reduction to the weighted graph (G, w) as follows. Define the vertex weight \bar{w} within $G[S^-]$ such that $\bar{w}(v) = d^-(v) - b^-(v)$ and $\bar{w}(u) = 1$, $\forall u \in N^-(v)$. Let $\rho = \min\{w(u)/\bar{w}(u) \mid u \in S^-\}$ and $S_0^- = \{u \in S^- \mid w(u) = \rho\bar{w}(u)\}$ so that $S_0^- \neq \emptyset$ and $w(u) - \rho\bar{w}(u) > 0$, $\forall u \in S^- - S_0^-$.

Suppose a solution C is computed for $G - S_0^- = G[V - S_0^-]$ under the weight $w - \rho\bar{w}$ defined on $V - S_0^-$. Then, $C \cup S_0^-$ is a solution for G and our algorithm

returns it. The ratio of this solution to the optimum, local to $(G[S^-], \rho\bar{w})$, is bounded by

$$\begin{aligned} \frac{\bar{w}(S^- \cap (C \cup S_0^-))}{d^-(v) - b^-(v)} &\leq \frac{\bar{w}(S^-)}{d^-(v) - b^-(v)} \\ &= \frac{2d^-(v) - b^-(v)}{d^-(v) - b^-(v)} \\ &= 2 + \frac{1}{d^-(v)/b^-(v) - 1}. \end{aligned}$$

So, if C is a p -approximation for $(G - S_0^-, w - \rho\bar{w})$, the approximation ratio of $C \cup S_0^-$ for (G, w) can be estimated by the following bound

$$\max \left\{ p, 2 + \frac{1}{d^-(v)/b^-(v) - 1} \right\}. \quad (2)$$

Thus, we apply the local ratio approximation to $G[S^-]$ and reduce to the problem on $G - S_0^-$ when such a vertex is found whose in-degree is large enough relative to its degree bound. Likewise, we may apply the local ratio reduction to out-degrees, and for $v \in V$ with $d^+(v) > b^+(v)$ we have the approximation ratio of $C \cup S_0^+$ for (G, w) bounded by

$$\max \left\{ p, 2 + \frac{1}{d^+(v)/b^+(v) - 1} \right\} \quad (3)$$

when $C \subseteq V - S_0^+$ is a p -approximation for the reduced problem on $G - S_0^+$. We apply these local ratio reductions as long as there remains a vertex v with high degree/degree-bound ratio; i.e., any v with $d^-(v)/b^-(v)$ or $d^+(v)/b^+(v)$ exceeding the threshold β .

2nd stage. We switch to the greedy algorithm for SSC (V, f, w) when vertices with high degree/degree-bound ratio are exhausted in the 1st stage. Here in the greedy mode, a vertex v with minimum $w(v)/f_C(\{v\})$ among the remaining vertices is repeatedly added to a solution set C as long as $f(C) < f(V)$.

We can show the following performance of this algorithm (details are omitted due to space limitations).

Theorem 2. *The (b^-, b^+) -BDD problem can be approximated within a factor of $2 + \max_{v \in V} \ln(b^-(v) + b^+(v))$ if $V_0 = \emptyset$.*

4 Partially Bounded Degree Deletion

4.1 Approximation Hardness

As was seen in the previous section, the Okun-Barak algorithm or its extension to directed graphs yields an $O(\log b_{\max})$ -approximation. We observe here that such performance is possible only when all the degrees, both in-degrees and

out-degrees, are bounded, and if not, even at a single vertex, the performance becomes unbounded even if b_{\max} is a fixed constant.

As already seen, the algorithm of Okun and Barak attains the best approximation bound of $2 + \ln b_{\max}$ for general b , by first applying the local ratio reduction to any v and its neighbors having high $d(v)$ to $b(v)$ ratio, and then by running the greedy approximation after $d(v)/b(v)$ becomes small enough for all the remaining vertices v . This approach is possible only when all the degrees are bounded since, if $d(v)$ is not bounded for some $v \in V$, there is now way of doing the local reduction at or around v with a reasonable ratio. Consider the following instance, for example: Let $G_b = (V_b, E_b)$ be a $(b - 1)$ -regular graph on n vertices, and V_b^c be a copy of V_b . Construct a graph $G = \{V_b \cup V_b^c \cup \{s\}, E\}$ from G_b by, besides having E_b entirely, connecting each vertex of V_b and its copy in V_b^c by an edge, and by having one more vertex s connected with every vertex in V_b by an edge. Since $d(v) = b + 1$ if $v \in V_b$, $= 1$ if $v \in V_b^c$, and $= n$ if $v = s$, when the degree bound is set s.t. $b(v) = b$, $\forall v \in V_b$ and $b(v) = 1$, $\forall v \in V_b^c$ (and the degree of s is unbounded), the $d(v)/b(v)$ ratio can be made arbitrarily close to 1 at every $v \in V_b \cup V_b^c$, that there is nowhere to apply the local ratio reduction. So the algorithm simply runs the standard greedy approximation to G . Suppose that all the vertices in V_b are of heavy weight while the vertices in V_b^c are respectively assigned with weights of $1, 1/2, 1/3, \dots, 1/n$ and s is assigned with $1 + \epsilon$. Then, the greedy algorithm outputs V_b^c as a solution of which weight is $\Theta(\log n)$ times that of the optimal solution $\{s\}$.

A more general approximation hardness of PBDD can be derived from that of Ek -VERTEX COVER ($EkVC$). This is the VERTEX COVER problem on k -uniform hypergraphs, and it is known to be NP-hard to approximate $EkVC$ within a factor of $k - 1 - \epsilon$ for any $\epsilon > 0$ and $k \geq 3$ [12]. Let $H = (V, E_H)$ denote an instance of $EkVC$, i.e., a k -uniform hypergraph. Construct a bipartite instance $G = (V \cup E_H, E)$ of undirected PBDD from H s.t. $\{v, e_H\} \in E$, where $v \in V$ and $e_H \in E_H$, iff $v \in e_H$ in H . Set the weight of each vertex in E_H heavy enough that forces choice of vertices only from V and not from E_H . Notice that $d(e_H) = k$, $\forall e_H \in E_H$, and $EkVC$ is reduced to undirected PBDD by setting the degree bound of $k - 1$ on each of them while leaving all the others (in V) unbounded. It follows from the approximation hardness of $EkVC$ that

Theorem 3. *It is NP-hard to approximate PBDD, directed or undirected, within a factor of $b_{\max} - \epsilon$ for any $\epsilon > 0$ and $b_{\max} \geq 2$.*

4.2 Approximation Algorithm

Let us turn to an upper bound in approximation of b -PBDD, and next is a key lemma here:

Lemma 1. *For any minimal b -PBDD solution $X \subseteq V \cup V_0$ in $G = (V \cup V_0, E)$*

$$\sum_{v \in X} r^d(\delta(v)) \leq \max\{2, b_{\max} + 1\} r^d(E).$$

Proof. Omitted due to space limitations. \square

It is immediate from Corollary 1 and Lemma 1 that

Theorem 4. *The b -PBDD problem can be approximated within $\max\{2, b_{\max} + 1\}$.*

The bound of $\max\{2, b_{\max} + 1\}$ given in Lemma 1 or Theorem 4 is tight even if $V_0 = \emptyset$. Suppose a graph G consists of the vertex set $X \cup Y \cup \{z\}$ and the edge set $E = X \times (Y \cup \{z\})$ s.t. $b(v) = 0, \forall v \in X \cup \{z\}$ and $b(v) = b_{\max}, \forall v \in Y$ for some integer b_{\max} . Clearly, X here is a minimal solution for b -PBDD.

Let x and y denote $|X|$ and $|Y|$, respectively. We have

$$r^d(E) = |E| - b(V) = x(y + 1) - b_{\max}y = (x - b_{\max})y + x$$

and

$$\sum_{v \in X} r^d(\delta(v)) = x(y + 1) = (x - b_{\max})y + x + b_{\max}y$$

since $r^d(\delta(v)) = y + 1, \forall v \in X$. Set $x = b_{\max} + 1$. Then,

$$\frac{\sum_{v \in X} r^d(\delta(v))}{r^d(E)} = \frac{x + y + b_{\max}y}{x + y} = 1 + \frac{b_{\max}y}{y + b_{\max} + 1}$$

and $\sum_{v \in X} r^d(\delta(v))/r^d(E)$ becomes arbitrarily close to $1 + b_{\max}$ as $y \rightarrow \infty$.

Suppose now that each vertex of G is weighted s.t. $w(v) = r^d(\delta(v))$. The algorithm PD may return X as an approximate solution, whose weight is $(b_{\max} + 1)(y + 1)$, whereas $\{v, z\}$ is an optimal solution for any $v \in X$ when y is large enough, whose weight is $d(v) + d(z) = y + b_{\max} + 2$. Therefore, the ratio of weight of X to the optimal weight is

$$\frac{(b_{\max} + 1)(y + 1)}{y + b_{\max} + 2} = 1 + b_{\max} - \frac{b_{\max}^2 + 2b_{\max} + 1}{y + b_{\max} + 2}$$

and it approaches arbitrarily close to $1 + b_{\max}$ as y becomes larger.

References

1. Balasundaram, B., Butenko, S., Hicks, I.V.: Clique relaxations in social network analysis: the maximum k -plex problem. *Oper. Res.* **59**(1), 133–142 (2011)
2. Bar-Yehuda, R.: Using homogeneous weights for approximating the partial cover problem. *J. Algorithms* **39**(2), 137–144 (2001)
3. Betzler, N., Bredereck, R., Niedermeier, R., Uhlmann, J.: On bounded-degree vertex deletion parameterized by treewidth. *Discret. Appl. Math.* **160**(1–2), 53–60 (2012)
4. Brešar, B., Krivoš-Belluš, R., Semanišin, G., Šparl, P.: On the weighted k -path vertex cover problem. *Discret. Appl. Math.* **177**, 14–18 (2014)
5. Brešar, B., Jakovac, M., Katrenič, J., Semanišin, G., Taranenko, A.: On the vertex k -path cover. *Discret. Appl. Math.* **161**(13–14), 1943–1949 (2013)

6. Brešar, B., Kardoš, F., Katrenič, J., Semanišin, G.: Minimum k -path vertex cover. *Discret. Appl. Math.* **159**(12), 1189–1195 (2011)
7. Camby, E., Cardinal, J., Chapelle, M., Fiorini, S., Joret, G.: A primal-dual 3-approximation algorithm for hitting 4-vertex paths. In: 9th International Colloquium on Graph Theory and Combinatorics, ICGT 2014, p. 61 (2014)
8. Case, B.M., Hedetniemi, S.T., Laskar, R.C., Lipman, D.J.: Partial domination in graphs. *arXiv e-prints* (2017)
9. Chauve, C., Tannier, E.: A methodological framework for the reconstruction of contiguous regions of ancestral genomes and its application to mammalian genomes. *PLoS Comput. Biol.* **4**(11), e1000234 (2008)
10. Chen, Z.-Z., Fellows, M., Fu, B., Jiang, H., Liu, Y., Wang, L., Zhu, B.: A linear kernel for co-path/cycle packing. In: Chen, B. (ed.) *AAIM 2010*. LNCS, vol. 6124, pp. 90–102. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14355-7_10
11. Das, A.: Partial domination in graphs. *arXiv e-prints* (2017)
12. Dinur, I., Guruswami, V., Khot, S., Regev, O.: A new multilayered PCP and the hardness of hypergraph vertex cover. *SIAM J. Comput.* **34**(5), 1129–1146 (2005)
13. Dinur, I., Safra, S.: On the hardness of approximating minimum vertex cover. *Ann. Math. (2)* **162**(1), 439–485 (2005)
14. Elomaa, T., Kujala, J.: Covering analysis of the greedy algorithm for partial cover. In: Elomaa, T., Mannila, H., Orponen, P. (eds.) *Algorithms and Applications*. LNCS, vol. 6060, pp. 102–113. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12476-1_7
15. Fellows, M.R., Guo, J., Moser, H., Niedermeier, R.: A generalization of Nemhauser and Trotter’s local optimization theorem. *J. Comput. Syst. Sci.* **77**(6), 1141–1158 (2011)
16. Feng, Q., Wang, J., Li, S., Chen, J.: Randomized parameterized algorithms for P_2 -packing and co-path packing problems. *J. Comb. Optim.* **29**(1), 125–140 (2015)
17. Fujito, T.: A unified approximation algorithm for node-deletion problems. *Discret. Appl. Math.* **86**(2–3), 213–231 (1998)
18. Fujito, T.: On approximation of the submodular set cover problem. *Oper. Res. Lett.* **25**(4), 169–174 (1999)
19. Fujito, T.: Approximating bounded degree deletion via matroid matching. In: Fotakis, D., Pagourtzis, A., Paschos, V.T. (eds.) *CIAC 2017*. LNCS, vol. 10236, pp. 234–246. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57586-5_20
20. Gandhi, R., Khuller, S., Srinivasan, A.: Approximation algorithms for partial covering problems. *J. Algorithms* **53**(1), 55–84 (2004)
21. Halperin, E., Srinivasan, A.: Improved approximation algorithms for the partial vertex cover problem. In: Jansen, K., Leonardi, S., Vazirani, V. (eds.) *APPROX 2002*. LNCS, vol. 2462, pp. 161–174. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45753-4_15
22. Jakovac, M., Taranenko, A.: On the k -path vertex cover of some graph products. *Discret. Math.* **313**(1), 94–100 (2013)
23. Karakostas, G.: A better approximation ratio for the vertex cover problem. *ACM Trans. Algorithms* **5**(4), art. no. 41 (2009)
24. Kardoš, F., Katrenič, J., Schiermeyer, I.: On computing the minimum 3-path vertex cover and dissociation number of graphs. *Theoret. Comput. Sci.* **412**(50), 7009–7017 (2011)
25. Khot, S., Regev, O.: Vertex cover might be hard to approximate to within $2 - \epsilon$. *J. Comput. Syst. Sci.* **74**(3), 335–349 (2008)

26. Kneis, J., Mölle, D., Rossmanith, P.: Partial vs. complete domination: t -dominating set. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 367–376. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-69507-3_31
27. Lee, E.: Partitioning a graph into small pieces with applications to path transversal. In: Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, pp. 1546–1558 (2017)
28. Moser, H., Niedermeier, R., Sorge, M.J.: Exact combinatorial algorithms and experiments for finding maximum k -plexes. *J. Comb. Optim.* **24**(3), 347–373 (2012)
29. Newman, I., Sohler, C.: Every property of hyperfinite graphs is testable. *SIAM J. Comput.* **42**(3), 1095–1112 (2013)
30. Okun, M., Barak, A.: A new approach for approximating node deletion problems. *Inform. Process. Lett.* **88**(5), 231–236 (2003)
31. Orlovich, Y., Dolgui, A., Finke, G., Gordon, V., Werner, F.: The complexity of dissociation set problems in graphs. *Discret. Appl. Math.* **159**(13), 1352–1366 (2011)
32. Seidman, S.B., Foster, B.L.: A graph-theoretic generalization of the clique concept. *J. Math. Soc.* **6**(1), 139–154 (1978)
33. Slavík, P.: Improved performance of the greedy algorithm for partial cover. *Inform. Process. Lett.* **64**(5), 251–254 (1997)
34. Tu, J.: A fixed-parameter algorithm for the vertex cover P_3 problem. *Inform. Process. Lett.* **115**(2), 96–99 (2015)
35. Tu, J., Yang, F.: The vertex cover P_3 problem in cubic graphs. *Inform. Process. Lett.* **113**(13), 481–485 (2013)
36. Tu, J., Zhou, W.: A factor 2 approximation algorithm for the vertex cover P_3 problem. *Inform. Process. Lett.* **111**(14), 683–686 (2011)
37. Tu, J., Zhou, W.: A primal-dual approximation algorithm for the vertex cover P_3 problem. *Theoret. Comput. Sci.* **412**(50), 7044–7048 (2011)
38. Wolsey, L.A.: An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica* **2**(4), 385–393 (1982)
39. Xiao, M.: On a generalization of Nemhauser and Trotter’s local optimization theorem. *J. Comput. Syst. Sci.* **84**, 97–106 (2017)
40. Yannakakis, M.: Node-deletion problems on bipartite graphs. *SIAM J. Comput.* **10**(2), 310–327 (1981)

Minimum-Width Annulus with Outliers: Circular, Square, and Rectangular Cases

Hee-Kap Ahn¹ , Taehoon Ahn¹, Sang Won Bae² , Jongmin Choi¹,
Mincheol Kim¹, Eunjin Oh¹, Chan-Su Shin³, and Sang Duk Yoon¹

¹ Department of Computer Science and Engineering, POSTECH,
Pohang, South Korea

{heekap,sloth,icothos,rucatia,jin9082,egooana}@postech.ac.kr

² Department of Computer Science, Kyonggi University, Suwon, South Korea
swbae@kgu.ac.kr

³ Division of Computer and Electronic Systems Engineering,
Hankuk University of Foreign Studies, Yongin, South Korea
cssin@hufs.ac.kr

Abstract. We study the problem of computing a minimum-width annulus with outliers. Specifically, given a set of n points in the plane and a nonnegative integer $k \leq n$, the problem asks to find a minimum-width annulus that contains at least $n - k$ input points. The k excluded points are considered as outliers of the input points. In this paper, we are interested in particular in annuli of three different shapes: circular, square, and rectangular annuli. For the three cases, we present first and improved algorithms to the problem.

1 Introduction

An annulus is a region bounded by two concentric circles. There are a few applications of computing the minimum annulus enclosing a set of points in the plane. For instance, one of the topics in metrology is to measure the roundness of an object, which is done mostly by measuring points obtained from the boundary of the object. If the width of the annulus that covers the measured points is close to zero or below a predefined threshold, then one can say that the object is (almost) round. Otherwise, the object is not round enough, and therefore it should be rejected. Another application is to locate an obnoxious or undesirable facility in a set of sites that use or get served by the facility in the plane. No one wants to have an obnoxious facility such as a garbage dump in his/her backyard but it should be located within a reasonable distance from the sites. A good

H.-K. Ahn, T. Ahn, J. Choi, M. Kim, E. Oh, and S.D. Yoon were supported by the MSIT (Ministry of Science and ICT), Korea, under the SW Starlab support program (IITP-2017-0-00905) supervised by the IITP (Institute for Information & communications Technology Promotion). S.W. Bae was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2015R1D1A1A01057220). C.-S. Shin was supported by University Research Grant of Hankuk University of Foreign Studies.

location for such an obnoxious facility is the one whose closest site is far enough and whose farthest site is not too far.

The *minimum-width* annulus that encloses a set P of points reflects the roundness of the point set P well. There has been a fair amount of work on the minimum-width annulus for a set of points in the plane. However, the data we obtain in applications often contains outliers which are due to variability in the measurement or errors in transmission. Outliers can be seen as violation of constraints in the minimum-width annulus problem: the points in P are to be covered by the annulus but some of them are allowed to be violated. In this paper, we study the minimum-width annulus problem for points containing outliers in the plane.

The minimum-width annulus problem has been studied in computational geometry. Motivated by the roundness test in metrology, Ebara et al. [9] presented a simple quadratic time algorithm that computes a minimum-width circular annulus enclosing a given set of points in the plane using Voronoi diagrams. Later, Agarwal et al. [4] presented an algorithm that uses Megiddo's parametric search technique and computes the minimum-width circular annulus in $O(n^{8/5+\varepsilon})$ time for any $\varepsilon > 0$. The problem was reconsidered by Agarwal et al. [2] as an application of computing the vertices, edges and 2-dimensional faces of the lower envelopes of multivariate functions. Their algorithm takes $O(n^{17/11+\varepsilon})$ expected time for any $\varepsilon > 0$. Then, Agarwal and Sharir [3] simplified and improved their previous algorithm by using vertical decomposition to $O(n^{3/2+\varepsilon})$ expected time for any $\varepsilon > 0$. Chan gave an $(1+\varepsilon)$ -approximation algorithm for the problem [8]. However, there is no algorithm known for the problem in the presence of outliers, except an approximation algorithm by Har-Peled and Wang [11].

There also has been research on variations of the minimum-width annulus problem depending on the *shape* of the annulus as well as the *distance metric* for measuring the width. A square or rectangular annulus is the region bounded by two concentric axis-parallel squares or rectangles, respectively. Abellanas et al. [1] presented an $O(n)$ -time algorithm for the rectangular annulus problem and considered several variations of the problem. Gluchshenko et al. [10] gave an optimal $O(n \log n)$ -time algorithm for a minimum-width square annulus that encloses n points in the plane. Later, Mukherjee et al. [14] presented an $O(n^2 \log n)$ -time algorithm that computes a minimum-width rectangular annulus over all orientations, and Bae [5] showed that a minimum-width square annulus over all orientations can be computed in $O(n^3 \log n)$ time.

In this paper, we study the problem of computing a minimum-width annulus that contains at least $n - k$ input points, when n points are given as input points and $k \leq n$ is also a part of input. The k excluded points are considered *outliers* of the n input points, and this problem is often called the minimum-width annulus problem with k outliers. We are interested in annuli of three different shapes: circular, square and rectangular annuli. See Fig. 1 for an illustration.

Very recently, Bae [6] considered the square or rectangular annulus problem with k outliers and presented several first algorithms. Among them, he presented an $O(k^2 n \log n + k^3 n)$ -time algorithm for the square annulus with $k \geq 1$ outliers

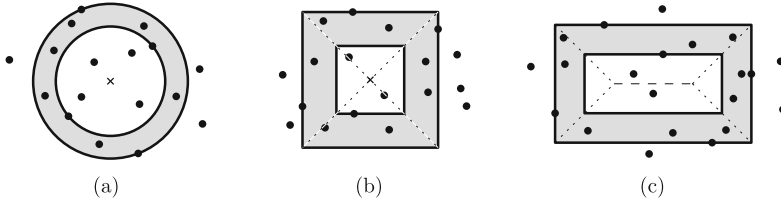


Fig. 1. Minimum-width (a) circular, (b) square, and (c) rectangular annuli with $k = 7$ outliers for a given set of points.

and $O(nk^2 \log k + k^4 \log^3 k)$ -time algorithm for the rectangular annulus with k outliers. It is worth noting that, when k is a constant, these running times match the lower bounds of the problems. On the other hand, no nontrivial algorithm for the minimum-width circular annulus with outliers is known so far.

Our results in this paper are threefold:

- We give an $O(k(kn)^{3/2+\varepsilon})$ -time algorithm for the minimum-width circular annulus with k outliers. This is the first nontrivial algorithm for the circular variant of the problem.
- We present an $O(k^2 n \log n)$ -time algorithm for the minimum-width square annulus problem with k outliers. This improves the previously best algorithm by Bae [6], which takes $O(k^2 n \log n + k^3 n)$ time.
- We also present two algorithms for the minimum-width rectangular annulus with k outliers whose running times are $O(n \log n + k^4 \log^2 n)$ and $O(nk^2 \log k + k^4 \log^2 k)$. Both of our algorithms are faster than the previously best known ones [6], which take $O(n \log^2 n + k^4 \log k \log^2 n)$ time and $O(nk^2 \log k + k^4 \log^3 k)$ time, respectively.

2 Preliminaries

In this paper, we are interested in annuli of three different shapes: circular, square, and rectangular annuli. A *circular annulus* is a closed region in the plane bounded by two concentric circles. The bigger circle that bounds a circular annulus A is called the *outer circle* of A , while the other is called the *inner circle* of A . The *width* of a circular annulus is the difference between the radii of its outer and inner circles.

For the square and rectangular cases, we only consider axis-parallel squares and rectangles. So, throughout the paper, any square or rectangle we discuss is supposed to be axis-parallel, unless stated otherwise. Consider a rectangle, or possibly a square, R in the plane \mathbb{R}^2 . We call the intersection point of its two diagonals the *center* of R .

An (*inward*) *offset* of R by $\delta > 0$ is a rectangle obtained by sliding the four sides of R inwards by δ . If the shorter side of R is of length r , then the offset of R by $\delta = \frac{1}{2}r$ is degenerated to a line segment or a point. For any positive $\delta \leq \frac{1}{2}r$, consider an inward offset R' of R by δ . Then, the closed region A between R

and R' , including its boundary, is called a *rectangular annulus* with the *outer rectangle* R and the *inner rectangle* R' . When R is a square and so is R' , the annulus A is called a *square annulus*, and R and R' are called its *outer square* and *inner square*, respectively. The distance δ between the sides of R and R' is called the *width* of the annulus.

Consider an annulus A , regardless of its shape. The complement $\mathbb{R}^2 \setminus A$ of the annulus A is separated into two connected components. We shall call the outside of its outer boundary the *outside* of A and the inside of its inner boundary the *inside* of A .

3 Circular Annulus with Outliers

In this section, we consider the problem of finding a minimum-width circular annulus with k outliers for a given set P of points. As observed in [2–4], the outer and inner circles of a minimum-width circular annulus are determined by four points of P : (1) one of its outer and inner circles has three points on it and the other has one, or (2) both have two points on each. This implies the following for minimum-width circular annuli with outliers.

Lemma 1. *There exists a minimum-width circular annulus A of P with k outliers such that one of the following conditions holds: (1) Three points in P lie on one of the inner and outer circles of A , and one point in P lies on the other circle. (2) Both the inner and outer circles of A have two points in P on each.*

Consider a minimum-width circular annulus A of P with k outliers that satisfies one of conditions (1) and (2) stated in Lemma 1. Let k_{in} be the number of points in P that lie in the inside of A and $k_{\text{out}} = k - k_{\text{in}}$ be the number of points in P that lie in the outside of A . In the following, we will show that the center of A is related to the *higher-order Voronoi diagrams* of P . The *order- t Voronoi diagram* of P , denoted by $\mathcal{V}_t(P)$, decomposes the plane \mathbb{R}^2 into Voronoi regions such that all points in each Voronoi region share the common t nearest points among those in P [12]. For more details on the order- t Voronoi diagrams, refer to Lee [12] and Liu et al. [13].

Lemma 2. *There exists a minimum-width circular annulus of P with k outliers such that its center lies on a vertex of the overlay of the order- $(k' + 1)$ Voronoi diagram $\mathcal{V}_{k'+1}(P)$ of P and the order- $(n - 1 - k + k')$ Voronoi diagram $\mathcal{V}_{n-1-k+k'}(P)$ for some $0 \leq k' \leq k$.*

This already yields a nontrivial algorithm: For each $0 \leq k' \leq k$, compute diagrams $\mathcal{V}_{k'+1}(P)$ and $\mathcal{V}_{n-1-k+k'}(P)$, compute the overlay of the two diagrams, and check every vertex of the overlay. Since the diagram $\mathcal{V}_t(P)$ has complexity $O(t(n-t))$ and can be computed in $O(t(n-t) \log n)$ time [13], we can compute a minimum-width circular annulus of P with k outliers roughly in time $O(k^3 n^2)$. In the following, we give a better solution.

Again, consider two cases stated in Lemma 1. As discussed above, in case (1), the center of our annulus A lies on a vertex of a higher-order Voronoi diagram. Thus, solutions falling into this case can be found without computing

the overlay. For the purpose, we compute the diagrams $\mathcal{V}_1, \mathcal{V}_2, \dots, \mathcal{V}_{k+1}$, and $\mathcal{V}_{n-1}, \mathcal{V}_{n-2}, \dots, \mathcal{V}_{n-k-1}$ in $O(k^2 n \log n)$ time [12, 13], and prepare each of the diagrams with a standard point location structure [7]. Then we are done by checking each vertex of $\mathcal{V}_{k'+1}(P)$ and its location on $\mathcal{V}_{n-1-k+k'}(P)$, taking $O(\log n)$ time per vertex. Hence, this case can be handled in total $O(k^2 n \log n)$ time.

Case (2) is relatively tricky. As above, consider a minimum-width circular annulus A of P with k outliers, and assume that this is case (2) and that k_{in} points in P lie in the inside of A and $k_{\text{out}} = k - k_{\text{in}}$ points in P lie in the outside of A . Here, we can directly extend the algorithm by Agarwal and Sharir [3] of computing the minimum-width annulus in $O(n^{3/2+\varepsilon})$ time as follows. In this case, as discussed above, each of the outer and inner circles passes through two points, thus its center lies on an edge of the corresponding higher-order Voronoi diagram. Using the lifting transformation that maps the points of P into the points on the paraboloid $z = x^2 + y^2$, a circle C with center (a, b) in the xy -plane is mapped to a plane $H(C)$ which is parallel to the plane tangent to the paraboloid at point $(a, b, a^2 + b^2)$. The intersection of $H(C)$ with the paraboloid is projected to the circle C in the xy -plane. Furthermore, a point lies on, inside, outside C if and only if its lifted point is on, below, above $H(C)$, respectively. Thus, if our annulus A misses k_{in} points in its inside and k_{out} points in its outside, then it is mapped to a pair of parallel planes such that k_{in} points are below the mapped plane of the inner circle and k_{out} points above the mapped plane of the outer circle.

A minimum-width annulus is transformed into a pair of two parallel planes that minimizes a (properly defined) distance function between the planes under the lifting [3]. In case (2), we observe that each of two mapped planes contains two lifted points, thus its projected circle has its center on the bisector of the two points, i.e., on an edge of the higher-order Voronoi diagram. By the same argument of Agarwal and Sharir [3], the problem of finding two parallel planes, each containing a line connecting two (lifted) points, with a minimum distance can be reduced to the problem of computing a closest pair of bichromatic lines in three dimension. This type of the closest line-pair problem can be solved in $O((|U| + |L|)^{3/2+\varepsilon})$ expected time for any positive $\varepsilon > 0$ by a randomized divide-and-conquer algorithm [3], where U is a set of the candidate lines contained in the upper plane (pairs of points lying on the outer circle) and L is a set of the candidate lines contained in the lower plane (pairs of points lying on the inner circle). Since U is obtained from the edges of the order- $(n - k_{\text{out}} - 1)$ Voronoi diagram $\mathcal{V}_{n-k_{\text{out}}-1}(P)$ of P and L is obtained from the edges of the order- $(k_{\text{in}}+1)$ Voronoi diagram $\mathcal{V}_{k_{\text{in}}+1}(P)$ of P , we have $|U| = O(k_{\text{out}}(n - k_{\text{out}})) = O(kn)$ and $|L| = O(k_{\text{in}}(n - k_{\text{in}})) = O(kn)$. Thus, it takes $O((kn)^{3/2+\varepsilon})$ expected time.

In order to handle case (2), we compute all those higher-order Voronoi diagrams, and for each $0 \leq k' \leq k$, we invoke the above algorithm for the closest line-pair problem. This correctly finds a minimum-width circular annulus with k outliers in case (2), and takes in total $O(k(kn)^{3/2+\varepsilon})$ expected time. Hence, we conclude the following theorem.

Theorem 1. *Given a set P of n points in the plane and a nonnegative integer $1 \leq k \leq n$, a minimum-width circular annulus of P with k outliers can be computed in $O(k(kn)^{3/2+\varepsilon})$ expected time for any $\varepsilon > 0$.*

4 Square Annulus with Outliers

In this section, we study the minimum-width square annulus problem with outliers, and present an $O(k^2 n \log n)$ -time algorithm that computes a minimum-width square annulus of a set P of n points with $k \leq n$ outliers. Throughout this section, for a point p , we denote by $x(p)$ and $y(p)$ the x -coordinate of p and the y -coordinate of p , respectively.

4.1 Configuration of Optimal Solutions

Bae [6] showed the following configuration of an optimal solution.

Lemma 3 (Bae [6]). *There exists a minimum-width square annulus of P with k outliers that contains two points in P lying on the opposite sides of its outer square.*

Moreover, the following lemma holds.

Lemma 4. *There exists a minimum-width square annulus of P with k outliers such that (1) one side of its inner square contains a point in P and three sides of its outer square contain points in P , or (2) two sides of its inner square contain points in P and two opposite sides of its outer square contain points in P .*

Consider a minimum-width square annulus of P with k outliers satisfying the condition in Lemma 4. We assume without loss of generality that both the left and right sides of its outer square contain points p_L and p_R in P , respectively. In the following, we describe how to find such an optimal solution, if any.

4.2 Finding Candidate Outer Squares

We observe that there are at most k points in P lying to the left of p_L . Similarly, there are at most k points lying to the right of p_R . To use this observation, we compute the set \mathcal{C} of pairs (p'_L, p'_R) of points in P such that there are at most k points in P lying to the left of p'_L and at most k points in P lying to the right of p'_R . Clearly, the size of \mathcal{C} is $O(k^2)$ and (p_L, p_R) is contained in \mathcal{C} .

In the following subsection, we present an algorithm that computes a minimum-width square annulus of P with k outliers in $O(n \log n)$ time, provided we are given p_L and p_R . To obtain a minimum-width square annulus, we apply this procedure with each pair of \mathcal{C} . Then we obtain $O(k^2)$ square annuli one of which is an optimal solution. We simply choose the one with smallest width.

4.3 Finding the Largest Inner Square for a Candidate Pair

Assume that we know p_L and p_R . We present an $O(n \log n)$ -time algorithm for computing a minimum-width square annulus of P with k outliers such that p_L lies on the left side of its outer square and p_R lies on the right side of its outer square.

Consider the squares whose left side contains p_L and whose right side contains p_R . All these squares have the same side length, that is, the difference of the x -coordinates of p_L and p_R , $x(p_R) - x(p_L)$. Moreover, the centers of such squares form a vertical line segment ℓ . Specifically, ℓ is the line segment that connects two points q_1 and q_2 such that: $x(q_1) = x(q_2) = x(p_L) + \rho$, $y(q_1) = \min\{y(p_L), y(p_R)\} + \rho$, and $y(q_2) = \max\{y(p_L), y(p_R)\} - \rho$, where $\rho = (x(p_R) - x(p_L))/2$. See Fig. 2(a) for an illustration.

For any point $t \in \ell$, we denote the square centered at t with side length $x(p_R) - x(p_L)$ by $S_{\text{out}}(t)$. By definition, it contains p_L and p_R on its left and right sides, respectively. We use $S_{\text{in}}(t, r)$ to denote the inner square centered at t with side length r . We know $r \leq x(p_R) - x(p_L)$, but do not know its exact value.

In the following, we find a largest possible inner square for a candidate pair (p_L, p_R) on the outer square whose corresponding annulus contains at least $n - k$ points. That is, we maximize $r \in [0, x(p_R) - x(p_L)]$ such that $S_{\text{out}}(t)$ and $S_{\text{in}}(t, r)$ form a square annulus that contains at least $n - k$ points in P for some $t \in \ell$. This determines the minimum-width square annulus for the fixed pair (p_L, p_R) .

For the purpose, we compute the set L of $O(n)$ candidate side lengths of inner square. We then apply a binary search on the sorted list L of candidate side lengths using the decision algorithm to find the interval I of two consecutive

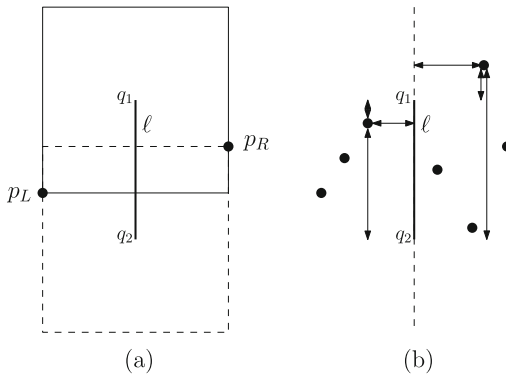


Fig. 2. (a) The set of centers of the squares with p_L and p_R on vertical sides constitutes a vertical line segment $\ell = q_1q_2$. (b) Candidate side lengths of inner squares with respect to ℓ . For each point p , its x -distance to ℓ and y -distance to each of the endpoints q_1 and q_2 of ℓ are candidate side lengths of inner squares. Precisely, $2|x(p) - x(\ell)|$, $2|y(p) - y(q_1)|$, and $2|y(p) - y(q_2)|$ are candidate side lengths of inner squares for a point $p \in P$.

side lengths in L containing the side length r^* of the largest inner square whose corresponding annulus contains at least $n - k$ points of P . To finally obtain r^* , we apply a linear search on another list of side lengths within I obtained by a series of certain events which will be described below.

Decision Algorithm. Assume that the points of P are sorted with respect to their y -coordinates. This allows us to sort the points in any subset P' of P with respect to their y -coordinates in $O(n)$ time. Let r be an input side length.

Imagine that a point t translates from one endpoint to the other endpoint of ℓ along ℓ . As t translates along ℓ , a point in P enters into $S_{\text{out}}(t)$ or $S_{\text{in}}(t, r)$, or exits from $S_{\text{out}}(t)$ or $S_{\text{in}}(t, r)$. We call a point $t \in \ell$ such that a point in P lies on the boundary of $S_{\text{out}}(t)$ or $S_{\text{in}}(t, r)$ an *event*. Note that for a point t lying between any two consecutive events along ℓ , the set $P \cap S_{\text{out}}(t)$ and the set $P \cap S_{\text{in}}(t, r)$ remain the same.

We compute all events and sort them along ℓ in $O(n)$ time. Then we translate t along ℓ and compute the number of points of P lying in the annulus with outer square $S_{\text{out}}(t)$ and inner square $S_{\text{in}}(t, r)$. Thus, we can determine whether there is some $t \in \ell$ such that the number of points of P lying in the square annulus determined by $S_{\text{out}}(t)$ and $S_{\text{in}}(t, r)$ is at least $n - k$ in $O(n)$ time in total.

Lemma 5. *Given $p_L, p_R \in P$ and $r > 0$, we can check in $O(n)$ time whether there is a square annulus of P with k outliers such that p_L lies on the left side of its outer square, p_R lies on the right side of its outer square, and its inner square has side length at most r for any input $r > 0$, provided that points in P are sorted with respect to their y -coordinates.*

Binary Search on Candidate Side Lengths. We first compute $O(n)$ candidate side lengths of the largest inner square whose corresponding annulus contains $n - k$ points of P . We then apply a binary search on the sorted list of these candidate side lengths. This gives us an interval I defined by two consecutive candidate side lengths in the sorted list containing the side length r^* of the largest inner square whose corresponding annulus contains $n - k$ points of P .

We consider the distance between each point $p \in P$ and the line containing ℓ , and take twice the value as a candidate side length. We also consider the difference between $y(p)$ for each $p \in P$ and $y(q)$ for each $q \in \{q_1, q_2\}$, and take twice the value as a candidate side length as well. (Recall that q_1 and q_2 are the two endpoints of ℓ .) Precisely, $2|x(p) - x(\ell)|$, $2|y(p) - y(q_1)|$, and $2|y(p) - y(q_2)|$ are candidate side lengths of inner squares for a point $p \in P$. See Fig. 2(b) for an illustration. We reject all distances larger than $x(p_R) - x(p_L)$ as no inner square of such a large side length defines a square annulus. We apply a binary search on the sorted list of all candidate side lengths we have taken using the decision algorithm above. As a result, we obtain the interval $I = [r', r'']$ bounded by two consecutive candidate side lengths, and I contains the side length r^* of the largest inner square whose corresponding annulus contains $n - k$ points of P .

Linear Search for r^* in I . Now we have the interval $I = [r', r'']$ containing the side length r^* of the largest inner square. For a value $r \in I$, consider a square S of side length r centered at a point $t \in \ell$. Imagine that we translate t from one endpoint of ℓ to the other endpoint (and therefore the square is translated accordingly.) Then the top side of the square hits a number of points in P . Observe that the set of points hit by the top side during the translation remains the same for any value $r \in I$ because there is no point p in P with $r' < 2|x(p) - x(\ell)| < r''$. This implies that the order of the points hit by the top side during the translation of the square remains the same even for varying $r \in I$. This also holds for the bottom side of the square.

For a fixed $r \in I$, consider all points p in P satisfying $|x(p) - x(\ell)| \leq r/2$ and $y(q_2) - r/2 \leq y(p) \leq y(q_1) + r/2$. They are the points in P that are swept by the moving inner square. Some of them are hit by the top or the bottom side during the translation. Let \mathcal{Q} be the list of these points sorted along the y -axis.

For a given $t \in \ell$, let $S_{\text{in}}(t)$ denote the inner square centered at t with largest side length in I that contains at least $n - k$ points in the annulus determined by $S_{\text{out}}(t)$ and $S_{\text{in}}(t)$. Let $P(t)$ denote the set of points in P contained in the annulus. Note that $S_{\text{in}}(t)$ may not be defined for a certain point $t \in \ell$ if the annulus determined by $S_{\text{out}}(t)$ and the square centered at t with side length r' contains less than $n - k$ points. In other words, $S_{\text{in}}(t)$ must have side length smaller than r' to contain at least $n - k$ points of P . Recall that r' is the smallest value in I . In this case, we let $S_{\text{in}}(t)$ denote the square centered at t with side length r' . Thus, the number of points in $P(t)$ is strictly smaller than $n - k$.

Now imagine that we translate t from the upper endpoint of ℓ to the lower endpoint. Then the set $P(t)$ changes at some points in ℓ during the translation. We call such a point on ℓ an *event*. We can characterize the events at which $P(t)$ may change during the translation as follows. To make the description easier, we assume that the annulus is open, that is, we consider the points of P lying on the boundary of the annulus as outliers.

Observation 1. *The set $P(t)$ changes only if either (1) a point appears on the boundary of $S_{\text{out}}(t)$, (2) a point appears on the top side and another point appears on the bottom side of $S_{\text{in}}(t)$ simultaneously, or (3) a point appears on the top or bottom side of $S_{\text{in}}(t)$ and the side length of $S_{\text{in}}(t)$ is the smallest value r' in I .*

The three cases correspond to three types of events, respectively. Clearly there are $O(n)$ events of the first type. Since the points in P are already sorted in their y -coordinates, we can compute the sorted list of the events of the first type in $O(n)$ time.

An event of the third type occurs at $t = y(p) + r'/2$ and $t = y(p) - r'/2$ for a point $p \in \mathcal{Q}$. As the points in P are already sorted in their y -coordinates, we can compute the sorted list of the events of the third type in $O(n)$ time. Let \mathcal{E} be the sorted list of the events of the first and the third types.

In the following, we will show that there are $O(n)$ events of the second type. We compute them in $O(n)$ time while we translate a point t along ℓ . During the translation, we maintain two pointers, one for the event immediately before t

and one for the event immediately after t in \mathcal{E} . We also maintain a counter that counts the number of points in $P(t)$. We also maintain the y -coordinates of the top and bottom sides of $S_{\text{in}}(t)$ in \mathcal{Q} during the translation of t .

Let e be the current event on ℓ of any type. We compute the event e' next to e along ℓ in $O(1)$ time as follows. Consider first the case that e' is an event of the first or third type. If e is an event of the first or third type, then both e and e' are in \mathcal{E} and e' is the event next to e in \mathcal{E} . Thus we can compute e' in constant time. If e is an event of the second type, then we can find the event e' that occurs immediately after e in \mathcal{E} in constant time as we maintain the location of e in \mathcal{E} . We also compute $S_{\text{in}}(e')$ and the number of points in $P(e')$ from the counter of $P(e)$ in constant time.

Now consider the case that e' is an event of the second type. There are two cases on the points p_t and p_b in \mathcal{Q} lying on the top and bottom sides of $S_{\text{in}}(e')$, respectively: (i) p_t lies on the top side of $S_{\text{in}}(e)$ and p_b is the first point in \mathcal{Q} lying below the bottom side of $S_{\text{in}}(e)$, or (ii) p_t is the first point in \mathcal{Q} lying below the top side of $S_{\text{in}}(e)$ and p_b is the first point in \mathcal{Q} lying below the bottom side of $S_{\text{in}}(e)$. Since we have the y -coordinates of the top and bottom sides of $S_{\text{in}}(e)$ in \mathcal{Q} , we can obtain e' and $S_{\text{in}}(e')$ in constant time. Moreover, the number of events of the second type is $O(n)$ in total because the bottom side of $S_{\text{in}}(t)$ moves downwards as t moves downwards along ℓ . We compute $S_{\text{in}}(e')$ and update the number of points of $P(e')$ in constant time.

Therefore, we can compute all $O(n)$ events in $O(n)$ time. By Lemma 4, one of the events is the center of a minimum-width annulus. Note that we obtain the number of points lying in the annulus for each event during the translation. We choose the one with the minimum width among the annuli containing at least $n - k$ points.

Theorem 2. *Given a set P of n points in the plane and a nonnegative integer k with $1 \leq k \leq n$, a minimum-width square annulus of P with k outliers can be computed in $O(k^2 n \log n)$ time.*

5 Rectangular Annulus with Outliers

In this section, we present two algorithms for computing a minimum-width rectangular annulus of a set P of n points with $k \leq n$ outliers.

Our algorithm is based on the following lemma given by Bae [6]. Due to this lemma, we can find $O(k^4)$ candidates of the outer rectangle of an optimal annulus as we did in Sect. 4.2.

Lemma 6 (Bae [6]). *There exists a minimum-width rectangular annulus of P with k outliers such that each side of its outer rectangle contains a point in P .*

5.1 Finding the Smallest-Width Annulus for a Fixed Outer Rectangle

We assume that we are given a data structure constructed on P that allows us to count the number of points of P lying on a query rectangle in $O(\log n)$

time [7]. Such a data structure can be constructed in $O(n \log n)$ time and has $O(n \log n)$ size. We also assume that we are given two balanced binary search trees constructed on P , one \mathcal{T}_x with respect to their x -coordinates and the other \mathcal{T}_y with respect to their y -coordinates.

Let R be a candidate outer rectangle. Our goal in this subsection is to find the minimum-width annulus whose outer rectangle is R . In other words, we find the inner rectangle with respect to R containing $k - k_{\text{out}}$ points of P , where k_{out} is the number of points in P lying outside of R . Recall that a rectangular annulus is determined by its outer and inner rectangles and the inner rectangle is an inward offset of the outer rectangle.

Given a value $\delta \geq 0$, we can determine in $O(\log n)$ time whether the annulus with outer rectangle R of width at most δ contains at least $n - k$ points by checking whether at most $k - k_{\text{out}}$ points of P lie on the inward offset of R by δ using the data structure for counting queries. This is our decision algorithm for a fixed width $\delta \geq 0$.

Let δ^* be the minimum width such that our decision algorithm returns a positive answer, and R^* be the inward offset of R by δ^* . That is, the annulus determined by R and R^* is the optimal solution for fixed outer rectangle R , and its width is δ^* . To reduce the search space for δ^* , we make use of the observation that at least one side of R^* contains a point of P . Consider the case where the left side of R^* contains a point $p^* \in P$. Let x_1 be the x -coordinate of the left side of R and x_2 be the x -coordinate of the center of R . Then, it is obvious that the x -coordinate of p^* lies in the interval $[x_1, x_2]$.

Now, we are ready to describe our algorithm to find p^* and δ^* . It starts with two standard queries for x_1 and x_2 on the balanced binary search tree \mathcal{T}_x on P with respect to the x -coordinates, resulting in two paths from the root to a leaf in \mathcal{T}_x . The two search paths share a common part from the root and then split at some node v of \mathcal{T}_x . We traverse \mathcal{T}_x again from the split node v . By the construction, the x -coordinate x_v corresponding to v lies in $[x_1, x_2]$. We then apply our decision algorithm for $\delta = x_v - x_1$. If the result is positive, then we proceed to the left child of v ; otherwise, if negative, then we proceed to the right child of v . We apply our decision algorithm for this next node repeatedly until we reach a leaf of \mathcal{T}_x . Then, the leaf node corresponds to the point p^* , in this case. Since the height of \mathcal{T}_x is $O(\log n)$ and our decision algorithm takes $O(\log n)$ time, this procedure terminates in $O(\log^2 n)$ time.

The other cases, where p^* lies on the right, top, or bottom side of R^* , can be handled in a symmetric way by traversing the binary search tree on P with respect to the x -coordinates or y -coordinates.

5.2 Putting it all Together

Since we have $O(k^4)$ candidate outer rectangles by Lemma 6, we obtain the following by the above discussion.

Theorem 3. *Given a set P of n points in the plane and a nonnegative integer $k \leq n$, a minimum-width rectangular annulus of P with k outliers can be computed in $O(n \log n + k^4 \log^2 n)$ time.*

The time bound in Theorem 3 has a term of $n \log n$, and this does not match the case of $k = 0$ in which one can solve the problem in $O(n)$ time [1]. In order to reduce the running time for small k , we exploit the approach by Bae [6].

A subset $K \subseteq P$ is called a *kernel* for P if a minimum-width rectangular annulus of K with k outliers is also a minimum-width rectangular annulus of P with k outliers at the same time. Bae [6] presented a procedure to compute a kernel K of size $O(k^4)$ in $O(nk^2 \log k + k^4)$ time. After computing such a kernel K , we compute a minimum-width rectangular annulus of K with k outliers in $O(k^4 \log^2 k)$ time using Theorem 3. Hence, we conclude the following theorem.

Theorem 4. *Given a set P of n points in the plane and an integer $1 \leq k \leq n$, a minimum-width rectangular annulus of P with k outliers can be computed in $O(nk^2 \log k + k^4 \log^2 k)$ time.*

References

1. Abellanas, M., Hurtado, F., Icking, C., Ma, L., Palop, B., Ramos, P.: Best fitting rectangles. In: Proceedings of the European Workshop on Computational Geometry (EuroCG 2003), pp. 147–150 (2003)
2. Agarwal, P.K., Aronov, B., Sharir, M.: Computing envelopes in four dimensions with applications. *SIAM J. Comput.* **26**(6), 1714–1732 (1997)
3. Agarwal, P., Sharir, M.: Efficient randomized algorithms for some geometric optimization problems. *Discrete Comput. Geom.* **16**, 317–337 (1996)
4. Agarwal, P., Sharir, M., Toledo, S.: Applications of parametric searching in geometric optimization. *J. Algo.* **17**, 292–318 (1994)
5. Bae, S.W.: Computing a minimum-width square annulus in arbitrary orientation. In: Kaykobad, M., Petreschi, R. (eds.) WALCOM 2016. LNCS, vol. 9627, pp. 131–142. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30139-6_11
6. Bae, S.W.: Computing a minimum-width square or rectangular annulus with outliers. In: Dinh, T.N., Thai, M.T. (eds.) COCOON 2016. LNCS, vol. 9797, pp. 443–454. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42634-1_36
7. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: *Computational Geometry: Algorithms and Applications*. Springer, Heidelberg (2008). <https://doi.org/10.1007/978-3-540-77974-2>
8. Chan, T.: Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. *Int. J. Comput. Geom. Appl.* **12**, 67–85 (2002)
9. Ebara, H., Fukuyama, N., Nakano, H., Nakanishi, Y.: Roundness algorithms using the Voronoi diagrams. In: Abstracts 1st Canadian Conference on Computational Geometry (CCCG), p. 41 (1989)
10. Gluchshenko, O.N., Hamacher, H.W., Tamir, A.: An optimal $O(n \log n)$ algorithm for finding an enclosing planar rectilinear annulus of minimum width. *Oper. Res. Lett.* **37**(3), 168–170 (2009)
11. Har-Peled, S., Wang, Y.: Shape fitting with outliers. *SIAM J. Comput.* **33**(2), 269–285 (2004)
12. Lee, D.: On k -nearest neighbor Voronoi diagrams in the plane. *IEEE Trans. Comput.* **31**(6), 478–487 (1982)
13. Liu, C.H., Papadopoulou, E., Lee, D.T.: The k -nearest-neighbor Voronoi diagram revisited. *Algorithmica* **71**, 429–449 (2015)
14. Mukherjee, J., Mahapatra, P., Karmakar, A., Das, S.: Minimum-width rectangular annulus. *Theor. Comput. Sci.* **508**, 74–80 (2013)

Minimum-Width Square Annulus Intersecting Polygons

Hee-Kap Ahn[✉], Taehoon Ahn, Jongmin Choi, Mincheol Kim,
and Eunjin Oh[✉]

Pohang University of Science and Technology, Pohang, South Korea
{heekap,sloth,icothos,rucatia,jin9082}@postech.ac.kr

Abstract. For k (possibly overlapping) polygons of total complexity n in the plane, we present an algorithm for computing a minimum-width square annulus that intersects all input polygons in $O(n^2\alpha(n)\log^3 n)$ time, where $\alpha(\cdot)$ is the inverse Ackermann function. When input polygons are pairwise disjoint, the running time becomes $O(n\log^3 n)$. We also present an algorithm for computing a minimum-width square annulus for k convex polygons of total complexity n . The running times are $O(n\log k)$ for possibly overlapping convex polygons and $O(n + k\log n)$ for pairwise disjoint convex polygons.

1 Introduction

One of the fundamental optimization problems in computational geometry is to enclose or intersect input objects (such as points, line segments, and polygons) with a predefined geometric figure (such as a circle, a square, or a rectangle) of smallest size. There has been a significant amount of work for such enclosure and intersection problems [4, 7, 14, 15]. Given n points in the plane, Meggido [15] studied the problem of computing the smallest disk that encloses the points and presented an $O(n)$ -time algorithm for the problem. Later, Bhattacharya et al. [4] presented a linear-time algorithm for computing the radius of the smallest closed hypersphere that intersects all input hyperplanes in d -dimensional Euclidean space.

In this paper, we consider a variant of the intersection problem: Given k polygons of total complexity n in the plane, compute a minimum-width axis-parallel *square annulus* that intersects all input *polygons*. A square annulus is the region bounded by two squares centered at a common point, and its width is the half of the difference between the side lengths of the two squares.

The problem of computing the minimum-width circular annulus of a set of points has been extensively studied [1–3, 5]. The best known exact algorithm for this problem takes $O(n^{3/2+\varepsilon})$ time for any constant $\varepsilon > 0$. All these algorithms are based on the fact that the center of the minimum-width circular annulus is a vertex of the overlay of the nearest-point and farthest-point Voronoi diagrams

This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the SW Starlab support program (IITP–2017–0–00905) supervised by the IITP (Institute for Information & communications Technology Promotion.).

of the input points. Since the overlay has $\Theta(n^2)$ complexity in the worst case, they use methods that avoid computing the whole description of the overlay explicitly.

A minimum-width axis-parallel square annulus containing n input points can be computed in $O(n \log n)$ time [12]. A main observation used in the algorithm is that the complexity of the farthest-point Voronoi diagram of the input points under L_∞ -metric has a constant complexity. Therefore, the overlay of the nearest-point and farthest-point Voronoi diagrams has $O(n)$ complexity in this case.

The minimum-width axis-parallel rectangular annulus of a set of n points can be computed in $O(n)$ time [16] without using Voronoi diagrams. A main observation is that the outer rectangle of the minimum-width axis-parallel rectangular annulus is the smallest enclosing rectangle of the input points.

Our Results. Given k (possibly overlapping) polygons of total complexity n in the plane, we present an $O(n^2 \alpha(n) \log^3 n)$ -time algorithm for computing a minimum-width axis-parallel square annulus that intersects all input polygons, where $\alpha(\cdot)$ is the inverse Ackermann function. If the polygons are pairwise disjoint, we can solve the problem in $O(n \log^3 n)$ time.

We also consider the case that all input polygons are convex, and present an $O(n \log k)$ -time algorithm. When input polygons are pairwise disjoint, the algorithm takes $O(n + k \log n)$ time.

To our best knowledge, these are the first results on the problem.

Due to page limit, some proofs are removed. The missing proofs can be found in the full version of this paper.

2 Preliminaries

Let \mathcal{P} be a set of k polygons of total complexity n . A *minimum-width square annulus* of \mathcal{P} is an axis-parallel square annulus with the minimum width that intersects all polygons in \mathcal{P} . Figure 1 (Left) shows a minimum-width square annulus of six polygons. Throughout this paper, a square refers to an axis-parallel square. We assume the *general position condition* on \mathcal{P} that no two vertices of the polygons in \mathcal{P} have the same x - or y -coordinate. We can avoid this condition by slightly modifying our algorithm or applying a slight perturbation to the positions of the vertices [11]. For a finite set X , we use $|X|$ to denote the size of X .

We define two distance measures which we call the *outer-distance* and the *inner-distance*. For a point x and a polygon P in the plane, we define the *outer-distance* between x and P as the half of the side length of the smallest square centered at x that intersects P and denote it by $d_O(x, P)$. For a point x in the plane, we define the *inner-distance* between x and P as the half of the side length of the smallest square centered at x that encloses P and denote it by $d_I(x, P)$. See Fig. 1 (Right) for an illustration.

We consider two Voronoi diagrams of \mathcal{P} , one with respect to d_I and one with respect to d_O . For a point x in \mathbb{R}^2 , let $F_O(x)$ be the outer-distance between

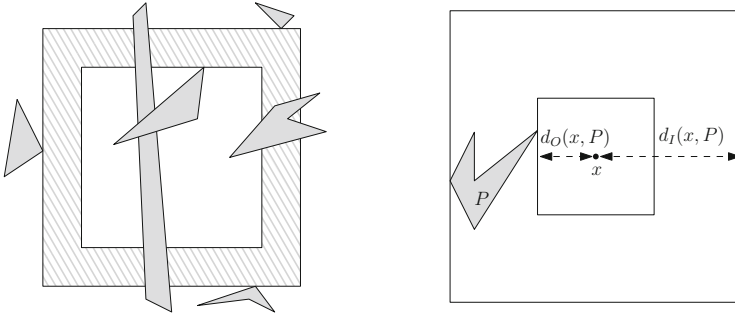


Fig. 1. Left: A minimum-width square annulus of six polygons. Right: $d_O(x, P)$ is the half of the side length of the smaller square and $d_I(x, P)$ is the half of the side length of the larger square.

x and the polygon of \mathcal{P} farthest from x under d_O . Similarly, let $F_I(x)$ be the inner-distance between x and the polygon of \mathcal{P} nearest to x under d_I . The xy -projections of $F_O(\cdot)$ and $F_I(\cdot)$ are the *farthest-outer Voronoi diagram* (denoted by VD_O) and the *nearest-inner Voronoi diagram* (denoted by VD_I).

The two Voronoi diagrams are subdivisions of \mathbb{R}^2 into cells, edges and vertices. We call an edge or a vertex of polygons in \mathcal{P} a *feature*. Each Voronoi cell of VD_O and VD_I corresponds to a feature e of a polygon P in \mathcal{P} . In this case, we say e defines the Voronoi cell. For VD_O , the feature e is nearest from a point x in the Voronoi cell among all features of P , and P is farthest from x among all input polygons under the outer-distance. Similarly, for VD_I , the feature e is farthest from a point x in the Voronoi cell among all features of P , and P is nearest from x among all input polygons under the inner-distance.

The following observation connects a minimum-width square annulus of \mathcal{P} and the farthest-outer and nearest-inner Voronoi diagrams.

Observation 1. *For any point x in the plane, the minimum-width square annulus of \mathcal{P} centered at x has width $\max\{F_O(x) - F_I(x), 0\}$.*

A *polyhedral terrain* is the graph of a piecewise linear continuous surface in three dimensions that intersects any line parallel to the z -axis in exactly one connected component (a point or a segment). The graph of $F_I(x)$ (and of $F_O(x)$) is a polyhedral terrain. We sometimes use F_I and F_O to denote the graphs of them if they are understood in context. By the observation above, our problem reduces to finding the minimum distance in z direction, that is, $\min_{x \in \mathbb{R}^2} F_O(x) - F_I(x)$, between the two polyhedral terrains.

Our overall strategy for computing a minimum-width square annulus of polygons is the following. We first compute two functions $F_I : \mathbb{R}^2 \rightarrow \mathbb{R}$ and $F_O : \mathbb{R}^2 \rightarrow \mathbb{R}$, and thus we obtain VD_I and VD_O . Then we find the minimum distance in z direction between F_I and F_O without constructing the overlay of VD_I and VD_O explicitly.

3 Minimum-Width Square Annulus for Polygons

We are given k polygons of total complexity n . We assume that each polygon is given as a list of vertices sorted along the polygon boundary. We present an algorithm for computing a minimum-width square annulus of the polygons. Our algorithm takes $O(n \log^3 n)$ time if the input polygons are pairwise disjoint, and $O(n^2 \alpha(n) \log^3 n)$ time otherwise, where $\alpha(\cdot)$ is the inverse Ackermann function.

3.1 Computing Voronoi Diagrams

The Farthest-Outer Voronoi Diagram. Consider the case that the input polygons are pairwise disjoint. Then the farthest-outer Voronoi diagram is the L_∞ -metric version of the Voronoi diagram studied by Cheong et al. [7]. They presented an $O(n \log^3 n)$ -time algorithm for computing the farthest-polygon Voronoi diagram of pairwise disjoint polygons of total complexity n , where the distance to a site (polygon) is measured by the Euclidean distance to a closest point on it. Since their algorithm also works for our case under the outer-distance, we can compute the farthest-outer Voronoi diagram of \mathcal{P} in the same time.

Now consider the general case that the input polygons are not necessarily pairwise disjoint. For a polygon $P \in \mathcal{P}$, consider the function that maps a point x in \mathbb{R}^2 to $d_O(x, P)$. The graph of this function is a polygonal terrain of complexity $|P|$, where $|P|$ denotes the complexity of P . By definition, F_O is the upper envelope of all such polygonal terrains over all polygons in \mathcal{P} .

Therefore, the problem reduces to computing the upper envelope of polygonal terrains of total complexity n , or more generally, computing the upper envelope of n triangles in \mathbb{R}^3 . Pach and Sharir [17] showed that the complexity of the upper envelope of n triangles in \mathbb{R}^3 is $O(n^2 \alpha(n))$. Edelsbrunner et al. [10] presented an algorithm for computing the upper envelope in $(n^2 \alpha(n))$ time. Therefore, we can compute the farthest-outer Voronoi diagram VD_O of \mathcal{P} in $O(n^2 \alpha(n))$ time.

The Nearest-Inner Voronoi Diagram. To compute VD_I , we use the algorithm for computing abstract Voronoi diagrams given by Klein [13]. Abstract Voronoi diagrams are based on systems of simple curves. The abstract Voronoi diagram of N sites has $O(N)$ complexity and can be computed in $O(N \log N)$ time if the family of bisecting curves is *admissible* [13]. For details, refer the paper [13].

In our case, we have k sites which are the polygons in \mathcal{P} . The bisecting curve between P and Q in \mathcal{P} is defined as the set of points in \mathbb{R}^2 equidistant from P and Q under the inner-distance. The following lemmas imply that the family of the bisecting curves is admissible. For a polygon P , let $R(P)$ denote the minimum enclosing axis-parallel rectangle of P .

Lemma 1. *The bisecting curve between any two (possibly overlapping) polygons P and P' coincides with the bisecting curve between $R(P)$ and $R(P')$.*

By Lemma 1, it suffices to consider the minimum enclosing axis-parallel rectangle $R(P)$ for each polygon P of \mathcal{P} instead of the original polygons. Consider the case that $R(P_1)$ for $P_1 \in \mathcal{P}$ contains $R(P_2)$ (and therefore contains P_2) for another polygon $P_2 \in \mathcal{P}$. In this case, there is no bisecting curve between P_1 and P_2 . Moreover, P_1 has no nonempty Voronoi cell. Thus, we remove all polygons $P \in \mathcal{P}$ such that $P' \subset R(P)$ for another polygon $P' \in \mathcal{P}$. We can find all such polygons in $O(k \log k)$ time by sweeping the plane with respect to a horizontal line once. Then we have the following lemma. The following lemma implies that the bisecting curve between any two polygons in \mathcal{P} is nonempty.

Lemma 2. *The bisecting curve between any two polygons consists of at most three edges, one is a line segment and the other two are rays, each of which is axis-parallel or has slope of ± 1 with respect to the x -axis.*

Lemmas 1 and 2 imply that the bisecting curve of any pair of two polygons P and P' of \mathcal{P} can be computed in constant time once we compute $R(P)$ and $R(P')$. Therefore, we have the following lemma.

Lemma 3. *For any two polygons of \mathcal{P} , we can compute their bisecting curve in constant time after processing the polygons in \mathcal{P} in $O(n)$ time.*

Therefore, the nearest-inner Voronoi diagram VD_I of k polygons with n vertices in total has $O(n)$ complexity and we can compute VD_I in $O(n + k \log k)$ time.

3.2 Searching the Region Between Two Polygonal Terrains

Our goal in this subsection is to find the minimum distance in z direction between two polygonal terrains F_I and F_O . In other words, we seek to find a point that minimizes $F_O(x) - F_I(x)$ over all points $x \in \mathbb{R}^2$. Notice that $F_O(x) - F_I(x)$ might be negative for a point x . In this case, the minimum-width square annulus of \mathcal{P} centered at x has width 0. Since the terrains are piecewise linear, the minimum distance is achieved at a vertex in the overlay of the two terrains.

Observation 2. *There is a vertex v in the overlay of VD_I and VD_O that gives the minimum value of $F_O(x) - F_I(x)$ over all points $x \in \mathbb{R}^2$.*

Thus a straightforward approach is to compute the overlay of VD_I and VD_O and to consider every vertex of the overlay as a candidate. Since the overlay is a plane graph with $O(nN)$ vertices, edges, and cells, it can be computed in $O(nN)$ time. Thus this approach can be done in $O(nN)$ time, where N is the complexity of VD_O . Recall that $N = O(n)$ if the polygons in \mathcal{P} are pairwise disjoint, and $N = O(n^2\alpha(n))$ otherwise.

There is an efficient algorithm for computing the minimum distance between two terrains without computing the overlay of them explicitly, and therefore achieving $o(nN)$ running time. Chazelle et al. [6] presented an algorithm that computes the minimum distance in z direction between any two polygonal terrains of total complexity N in $O(N^{4/3+\varepsilon})$ time for any constant $\varepsilon > 0$.

We may use their approach for our problem. However, in the following, we show how to improve the running time for our problem to $O(N \log^3 N)$. We use the approach in [1–3] together with a few new ideas to reduce the running time. Specifically, we make use of the fact that each edge of VD_I is axis-parallel or has slope of ± 1 . Due to this fact, we can encode an edge of F_I and F_O using only two variables. Note that in [1–3], each edge of Voronoi diagrams is encoded by four variables.

Consider a minimum-length segment ℓ parallel to the z -axis, one endpoint lying on F_I and the other endpoint lying on F_O .

There are two cases for the endpoints of ℓ [6]: (a) one endpoint is a vertex of F_I or F_O , or (b) one endpoint lies on an edge of F_I and the other lies on an edge of F_O . Case (a) can be handled as follows. For each vertex v of VD_I and VD_O , we compute $F_I(v)$ and $F_O(v)$ in $O(\log m)$ time, where m is the larger of the complexities of VD_I and VD_O . Thus we can handle case (a) in $O(m \log m)$ time.

To handle case (b), in the remaining of this section, we present data structures that allow us to find an edge e' of VD_O with smallest $F_O(v) - F_I(v)$ in $O(\log^3 m)$ time for each edge e of VD_I , where v is the intersection point between e and e' . By applying this procedure to every edge of VD_I , we can find a minimum-width square annulus of \mathcal{P} in $O(m \log^3 m)$ time excluding the time for computing F_I and F_O for case (b), where m is the larger of the complexities of VD_I and VD_O .

Pairs of Edge Subsets. We first compute, for each edge e of VD_I , the edges e' of VD_O intersecting e . Once we have the edges of VD_O intersecting e for each edge e of VD_I , we consider the line containing each such edge e' of VD_O , instead of e' .

To do this, we construct a set of edge pairs of VD_I and VD_O as follows. Consider a set \mathcal{E} of pairs (E, E') each of which consists of a subset E of the edge set of VD_I and a subset E' of the edge set of VD_O . We call \mathcal{E} an *IE-partition* if (1) every edge in E of each pair (E, E') of \mathcal{E} intersects every edge in E' and (2) there exists a pair $(E, E') \in \mathcal{E}$ for each $e \in E$ and $e' \in E'$ with $e \cap e' \neq \emptyset$. The *size* of an IE-partition is the sum of $|E| + |E'|$ over all pairs (E, E') of the IE-partition.

For the proof of the following lemma, see Sect. 4 of [6].

Lemma 4 ([6]). *An IE-partition of size $O(m \log^2 m)$ can be constructed in $O(m \log^2 m)$ time.*

Reduction to Ray-Shooting Towards a Lower Envelope. We compute an IE-partition \mathcal{E} of size $O(m \log^2 m)$. Then for each pair (E, E') of \mathcal{E} , we construct a data structure that allows us to find, for each (query) edge $e \in E$, an edge e' of E' with smallest $F_O(v) - F_I(v)$, where v is the intersection point between e and e' .

Recall that an edge of E is axis-parallel or has slope of ± 1 . We construct four data structures to handle these cases. In the following, we show how to handle the edges of E of slope 1. We can handle the other cases analogously.

Since every edge e in E intersects every edge in E' for a pair (E, E') , we simply treat each edge in $E \cup E'$ as the line containing the edge. We can encode a line e of E using two variables: its y -intercept and the coordinate of the feature defining e and lying below e . To see this, observe that e is a part of the bisector of two features, say p_1 and p_2 . One of them, say p_1 , lies below e . For any point x in e , the L_∞ -distance between x and p_1 is determined by only one coordinate of p_1 . Thus, to encode the distance between a point in e and p_1 , it suffices to use only one of two coordinates of p_1 . We map each edge $e \in E$ to the point $\mu(e)$ whose x - and y -coordinates are the y -coordinate of e and one coordinate of the feature defining e and lying below e , respectively.

For each edge $e' \in E'$, we consider a surface $\sigma_{e'}(\alpha, \beta, \gamma) = 0$ such that for every $e \in E$,

$$\sigma_{e'}(\alpha, \beta, F_O(v) - F_I(v)) = 0, \tag{1}$$

where $(\alpha, \beta) = \mu(e)$ and v is the intersection point between e and e' . Notice that such a surface is not unique.

Lemma 5. *There is a plane $\sigma_{e'}(\alpha, \beta, \gamma) = 0$ satisfying Eq. (1) for each edge e' in E' . Moreover, we can compute it in constant time for each edge in E' .*

For each edge $e' \in E'$, we use $\sigma_{e'}(\alpha, \beta, \gamma) = 0$ to denote the plane satisfying Eq. (1). The following lemma reduces the problem to the problem of vertical ray-shooting towards the lower envelope of the planes $\sigma_{e'}(\alpha, \beta, \gamma) = 0$ for all edges $e' \in E'$. This lemma holds by definition of $\sigma_{e'}(\alpha, \beta, \gamma) = 0$.

Lemma 6. *For an edge $e \in E$, let e' be an edge of E' such that $\sigma_{e'}(\alpha, \beta, \gamma) = 0$ is the lowest plane among all planes for the edges of E' intersected by the line passing through $\mu(e)$ and parallel to the z -axis. Then e' is the edge of E' with smallest $F_O(v) - F_I(v)$, where v is the intersection point of e and e' .*

Ray-Shooting Towards the Lower Envelope. We compute the lower envelope of all planes $\sigma_{e'}(\alpha, \beta, \gamma) = 0$ for all edges $e' \in E'$ in $O(|E'| \log |E'|)$ time [8], and then we find the face of the lower envelope intersecting the line parallel to the z -axis and passing through $\mu(e)$ for each edge $e \in E$ in $O(\log |E'|)$ time [18]. Then for each edge $e \in E$, we have an edge $e' \in E'$ with smallest $F_O(v) - F_I(v)$ by Lemma 6, where v is the intersection point of e and e' . We can do this in $O(|E| \log |E'| + |E'| \log |E'|)$ time for every edge e in E in total.

We do this for every pair of \mathcal{E} . Since the sum of $|E| + |E'|$ over all pairs (E, E') in \mathcal{E} is $O(m \log^2 m)$, the total running time is $O(m \log^3 m)$.

Theorem 1. *Given a set \mathcal{P} of polygons of total complexity n , a minimum-width square annulus of \mathcal{P} can be computed in $O(n^2 \alpha(n) \log^3 n)$ time. When the polygons are pairwise disjoint, the running time becomes $O(n \log^3 n)$.*

4 Minimum-Width Square Annulus for Convex Polygons

In this section, we consider the case that every polygon in \mathcal{P} is convex. We assume that each polygon is given as a list of vertices sorted along the polygon boundary. Our algorithm takes $O(n + k \log n)$ time if the polygons are pairwise disjoint, and $O(n \log k)$ time otherwise, where k is the number of the convex polygons of \mathcal{P} and n is the total complexity of the convex polygons.

The overall strategy is similar to the general case described in Sect. 3. A main observation for speeding up the algorithm is that the farthest-outer Voronoi diagram consists of $O(1)$ xy -monotone polygonal curves and $O(n)$ rays.

4.1 Properties of the Farthest-Outer Voronoi Diagram

We provide several combinatorial properties of the farthest-outer Voronoi diagram of convex polygons. Some of them are extensions of the properties for the case that sites are line segments which were given by Dey and Papadopoulou [9].

Definer of VD_O . We define the *definer* of VD_O as follows. We use it for analyzing combinatorial properties of the farthest-outer Voronoi diagram. Let x be a point in the Voronoi cell C corresponding to a feature e of a polygon $P \in \mathcal{P}$. We use $s(x)$ to denote a point (or an axis-parallel line segment) at which the smallest axis-parallel square centered at $x \in C$ and intersecting P intersects e . The definer of VD_O is the union of $s(x)$ for all points $x \in \mathbb{R}^2$.

Consider the case that $s(x)$ is in the top side (excluding its endpoints) of the smallest axis-parallel square centered at x and intersecting \mathcal{P} . Then P is the polygon which has the highest bottom side among all polygons in \mathcal{P} . Thus, there is only one feature e whose Voronoi cell belongs to this case. We denote the horizontal line passing through e by ℓ_t . This also holds for the other cases, that is, $s(x)$ lying on the bottom, left, and right sides, excluding their endpoints. In this way, we obtain four axis-parallel lines ℓ_i for $i = t, b, r, l$. (We use b for the bottom side, r for the right side, and l for the left side.)

Consider the case that $s(x)$ lies on a corner, say the top-right corner, of the smallest intersecting square of \mathcal{P} centered at x . The top-right corner lies in the north-east quadrant defined by ℓ_r and ℓ_t . However, note that x is not contained in the north-east quadrant. Otherwise, the smallest intersecting square of \mathcal{P} centered at x does not intersect the polygon defining at least one of ℓ_t, ℓ_b, ℓ_l and ℓ_r , which is a contradiction.

We claim that there is a convex polygonal curve γ_1 containing $s(x)$ for any $x \in \mathbb{R}^2$ belonging to the case that $s(x)$ lies on the top-right corner. For any polygon P in \mathcal{P} , the intersection of the boundary of P with the top-right quadrant defined by ℓ_r and ℓ_t consists of at most two boundary chains of P . This is simply because P is convex. If there are exactly two boundary chains of P in the intersection, $d_O(x, P)$ is determined by the chain of P closer to the the origin of the quadrant, that is, $d_O(x, P)$ coincides with the outer-distance between x and the convex chain. Otherwise, no feature of P belongs to this case.

Thus, by definition, $s(x)$ lies on the upper envelope of all such convex chains for all polygons in \mathcal{P} . We denote the upper envelope by γ_{tr} . The same holds for the other cases, that is, $s(x)$ is a corner other than the top-right corner, and we obtain four convex polygonal curves $\gamma_{tr}, \gamma_{br}, \gamma_{tl}$ and γ_{bl} .

Thus, the definer of VD_O consists of four features of the polygons in \mathcal{P} defining ℓ_t, ℓ_b, ℓ_l and ℓ_r each, and four convex polygonal curves $\gamma_{tr}, \gamma_{br}, \gamma_{tl}$ and γ_{bl} .

Properties of the Farthest-Outer Voronoi Diagram. Each edge of VD_O is either a line segment or a ray. We call an edge of the first type *bounded* and an edge of the second type *unbounded*. Recall that $F_O(x)$ is the outer-distance between x and the polygon in \mathcal{P} farthest from x for a point $x \in \mathbb{R}^2$. We say a function is convex if the region above its graph is a convex set.

Lemma 7. *Every cell in the farthest-outer Voronoi diagram is unbounded.*

Since the polygons are convex, the distance function d_O for each polygon is convex, and therefore $F_O(\cdot)$, the upper envelope of the graphs of d_O 's, is also convex.

Lemma 8. *The function $F_O : \mathbb{R}^2 \rightarrow \mathbb{R}$ is convex.*

Lemma 9. *There are $O(1)$ xy -monotone polygonal curves whose union contains every vertex and every bounded edge of VD_O .*

Lemma 10. *The farthest-outer Voronoi diagram of (possibly overlapping) convex polygons of total complexity n has complexity $O(n)$.*

Proof. We first consider the bounded edges and the vertices of VD_O . By Lemma 9, there are $O(1)$ xy -monotone polygonal curves whose union contains every vertex and every bounded edge of VD_O . Moreover, such xy -monotone polygonal curves have complexity of $O(n)$ in total since the complexity of the definer of VD_O is $O(n)$. Therefore, there are $O(n)$ bounded edges and vertices of VD_O .

Now consider the unbounded edges of VD_O . An unbounded edge of VD_O has one vertex of VD_O as its endpoint. Since there are $O(n)$ vertices of VD_O , the number of the unbounded edges of VD_O is $O(n)$.

Each cell is bounded by at least one edge of VD_O . Moreover, each edge of VD_O is incident to exactly two cells of VD_O . Therefore, the number of the cells of VD_O is $O(n)$, and the complexity of VD_O is $O(n)$. \square

4.2 Computing the Farthest-Outer Voronoi Diagram

Lemma 11. *The farthest-outer Voronoi diagram of k (possibly overlapping) convex polygons of total complexity n can be computed in $O(n \log k)$ time.*

Proof. We use a scheme similar to the one by Dey and Papadopoulou [9]. Recall that the definer of VD_O consists of four vertices of polygons in \mathcal{P} and four convex

chains. We can compute the four vertices in the definer of VD_O in $O(n)$ time as follows. For the vertex defining ℓ_t , we consider the polygons in \mathcal{P} one by one and find the lowest vertex of each polygon. Then we choose the highest one among all the lowest vertices, which defines ℓ_t . The other vertices of the definer of VD_O can be computed in a similar way.

We can compute the convex chain γ_{tr} in $O(n \log k)$ time as follows. For each polygon $P \in \mathcal{P}$ intersecting the top-right quadrant defined by ℓ_t and ℓ_r , we find the maximal convex chain of the intersection of the boundary of P with the quadrant. We can find all such convex chains for all polygons in \mathcal{P} in $O(n)$ time in total. Then we compute the upper envelope of the convex chains in $O(n \log k)$ time using divide-and-conquer technique. Therefore the total construction of the definer of VD_O takes $O(n \log k)$ time.

After constructing the definer of VD_O , we can construct VD_O in $O(n)$ time using the approach of Dey and Papadopoulou [9]. Once we have the definer of VD_O , we can compute the order of the Voronoi cells at infinity in $O(n)$ time. Then we sweep the plane towards the center of \mathcal{P} and compute VD_O in $O(n)$ time in total. Therefore, we can compute the farthest-outer Voronoi diagram of k possibly intersecting convex polygons in $O(n \log k)$ time in total. \square

For pairwise disjoint convex polygons, the upper envelopes can be computed in $O(n)$ time since each upper envelope is a part of a polygon boundary in \mathcal{P} .

Lemma 12. *The farthest-outer Voronoi diagram of k pairwise disjoint convex polygons of total complexity n can be computed in $O(n)$ time.*

4.3 Searching the Region Between Two Polygonal Terrains

As shown in Sect. 3, the problem of computing a minimum-width annulus of \mathcal{P} reduces to the problem of computing the minimum distance in z -direction between $F_O(x)$ and $F_I(x)$. In this case, $F_O(x)$ is convex by Lemma 8.

We can compute a minimum-width annulus of \mathcal{P} in $O(n \log n)$ time using the algorithm by Zhu [19]. The algorithm by Zhu computes the minimum distance in z -direction between two polyhedral terrains, one of which is convex in $O(N \log N)$ time, where N is the total complexity of $F_O(x)$ and $F_I(x)$.

We improve the running time for this procedure to $O(n + k \log n)$ using the structural properties of VD_O . Recall that there are two cases for endpoints of a minimum-length segment ℓ parallel to the z -axis: (a) one endpoint is a vertex of F_I or F_O , or (b) one endpoint lies on an edge of F_I and the other lies on an edge of F_O .

For the subcase of (a) that one endpoint is a vertex of F_I , we simply compute $F_O(v)$ for every vertex v in F_I in $O(\log n)$ time. Since there are $O(k)$ vertices of F_I , this takes $O(k \log n)$ time in total.

In the following, we consider the subcase of (a) that one endpoint is a vertex of F_O and case (b). Lemma 9 states that there are $O(1)$ xy -monotone polygonal curves containing all the bounded edges. Moreover, each vertex of VD_O lies on one of the xy -monotone polygonal curves. We first compute the intersection of

each xy -monotone polygonal curve with the edges of VD_I . We show that there are $O(k)$ intersection points which can be computed in $O(n + k \log k)$ time. Due to lack of space, we omit how we handle the latter case for the unbounded edges of VD_O . This can be found in the full version of the paper.

Bounded Edges and Vertices of VD_O . Due to Lemma 13, we can compute the intersection points of all edges of VD_I with the bounded edges of VD_O in total $O(n + k \log k)$ time. Also, we can compute $F_I(v)$ for each vertex v of F_O in $O(n + k \log k)$ time in total. To prove Lemma 13, we need the following observation.

Observation 3. *For any edge h of VD_I , the smallest axis-parallel rectangle containing h is not intersected by any other edge of VD_I .*

Lemma 13. *For a xy -monotone polygonal curve γ in the union of the bounded edges of VD_O , there are $O(n)$ intersection points of γ with the edges of VD_I . Moreover, we can compute the intersection points in $O(n + k \log k)$ time.*

Proof. We consider the vertical decomposition \mathcal{M} of VD_I obtained by drawing two vertical extensions from every endpoint of an edge of VD_I , one extension going upwards and one going downwards. The vertical decomposition \mathcal{M} is a finer subdivision of VD_I .

We claim that there are $O(k)$ intersection points of γ with the edges of \mathcal{M} . Without loss of generality, we assume that γ is *increasing*, that is, as we move a point in γ from one endpoint to the other endpoint, both x - and y -coordinates are increasing or decreasing.

Note that an edge of \mathcal{M} is axis-parallel or has slope of ± 1 . An axis-parallel edge of \mathcal{M} intersects γ at most once. An edge of slope -1 intersects γ at most once. Consider an edge e of slope 1. It is possible that γ intersects e more than once. However, we claim that the total number of intersection points for all edges of \mathcal{M} of slope 1 is $O(n)$. To see this, we observe that the part of γ lying between any two intersection points of e with γ is contained in the smallest axis-parallel rectangle containing e . This is because γ is xy -monotone. By Observation 3, the rectangle is not intersected by any other edge of VD_I (of \mathcal{M}). Therefore, the claim holds, and there are $O(n)$ intersection points of γ with the edges of VD_I .

Now, we present an algorithm for computing all intersection points. We compute \mathcal{M} in $O(k \log k)$ time [8]. Since \mathcal{M} is a trapezoidal decomposition of \mathbb{R}^2 of complexity $O(k)$, we can traverse the cells along γ in $O(n + K \log k)$ time, where K is the number of intersection points of the edges of \mathcal{M} parallel to the y -axis with γ . Since K is $O(k)$, we can compute the intersection points in $O(n + k \log k)$ time in total. \square

In summary, we can compute the minimum distance in z -direction between F_O and F_I in $O(k \log n)$ time once F_O and F_I are given.

Theorem 2. *Given a set \mathcal{P} of k convex polygons of total complexity n , a minimum-width square annulus of \mathcal{P} can be computed in $O(n \log k)$ time. When the polygons are pairwise disjoint, the running time becomes $O(n + k \log n)$.*

References

1. Agarwal, P.K., Aronov, B., Sharir, M.: Computing envelopes in four dimensions with applications. *SIAM J. Comput.* **26**(6), 1714–1732 (1997)
2. Agarwal, P.K., Sharir, M.: Efficient randomized algorithms for some geometric optimization problems. *Discrete Comput. Geom.* **16**(4), 317–337 (1996)
3. Agarwal, P.K., Sharir, M., Toledo, S.: Applications of parametric searching in geometric optimization. *J. Algorithms* **17**(3), 292–318 (1994)
4. Bhattacharya, B.K., Jadhav, S., Mukhopadhyay, A., Robert, J.M.: Optimal algorithms for some intersection radius problems. *Computing* **52**(3), 269–279 (1994)
5. Chan, T.M.: Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. *Int. J. Comput. Geom. Appl.* **12**(1–2), 67–85 (2002)
6. Chazelle, B., Edelsbrunner, H., Guibas, L.J., Sharir, M.: Algorithms for bichromatic line-segment problems and polyhedral terrains. *Algorithmica* **11**(2), 116–132 (1994)
7. Cheong, O., Everett, H., Glisse, M., Gudmundsson, J., Hornus, S., Lazard, S., Lee, M., Na, H.-S.: Farthest-polygon voronoi diagrams. *Comput. Geom.* **44**(4), 234–247 (2011)
8. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: *Computational Geometry: Algorithms and Applications*. Springer TELOS, Santa Clara (2008)
9. Dey, S.K., Papadopoulou, E.: The $L_\infty(L_1)$ farthest line-segment Voronoi diagram. In: Ninth International Symposium on Voronoi Diagrams in Science and Engineering, ISVD 2012, pp. 49–55 (2012)
10. Edelsbrunner, H., Guibas, L.J., Sharir, M.: The upper envelope of piecewise linear functions: algorithms and applications. *Discrete Comput. Geom.* **4**(4), 311–336 (1989)
11. Edelsbrunner, H., Mücke, E.P.: Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.* **9**(1), 66–104 (1990)
12. Gluchshenko, O.N., Hamacher, H.W., Tamir, A.: An optimal $O(n \log n)$ algorithm for finding an enclosing planar rectilinear annulus of minimum width. *Oper. Res. Lett.* **37**(3), 168–170 (2009)
13. Klein, R.: *Concrete and Abstract Voronoi Diagrams*. Springer, Heidelberg (1989). <https://doi.org/10.1007/3-540-52055-4>
14. Löffler, M., van Kreveld, M.: Largest bounding box, smallest diameter, and related problems on imprecise points. *Comput. Geom.* **43**(4), 419–433 (2010)
15. Megiddo, N.: Linear-time algorithms for linear programming in \mathbb{R}^3 and related problems. *SIAM J. Comput.* **12**(4), 759–776 (1983)
16. Mukherjee, J., Mahapatra, P.R.S., Karmakar, A., Das, S.: Minimum-width rectangular annulus. *Theoret. Comput. Sci.* **508**, 74–80 (2013)
17. Pach, J., Sharir, M.: The upper envelope of piecewise linear functions and the boundary of a region enclosed by convex plates: Combinatorial analysis. *Discrete Comput. Geom.* **4**, 291–309 (1989)
18. Sarnak, N., Tarjan, R.E.: Planar point location using persistent search trees. *Commun. ACM* **29**(7), 669–679 (1986)
19. Zhu, B.: Computing the shortest watchtower of a polyhedral terrain in $O(n \log n)$ time. *Comput. Geom.* **8**(4), 181–193 (1997)

Two New Schemes in the Bitprobe Model

Mirza Galib Anwarul Husain Baig and Deepanjan Kesh^(✉)

Indian Institute of Technology Guwahati, Guwahati, India
{mirza.baig,deepkesh}@iitg.ernet.in

Abstract. In this paper, we describe two new explicit schemes in the bitprobe model that, to the best of our knowledge, improves upon the existing schemes in the literature. One such scheme is to store three elements using two queries in the adaptive bitprobe model. Previously, the maximum number of elements that can be handled using two queries in the adaptive model is two, which is due to Radhakrishnan *et al.* [2]. A corollary of this result is an explicit scheme for storing three elements using three queries in the non-adaptive model, a first such scheme in the model. The second scheme is to store four elements using four queries in the non-adaptive bitprobe model. The previous scheme to store such a configuration was a non-explicit scheme due to Alon and Fiege [1], and we provide an explicit scheme for the same.

1 Introduction

Consider an universe \mathcal{U} of m elements. Further, consider a subset \mathcal{S} of \mathcal{U} containing n elements. In the *bitprobe model*, we study the problem of storing the subset \mathcal{S} in a datastructure of size s such that membership queries can be answered by probing at most t bits of the datastructure. For a given m and n , the schemes in this model try to optimise the space used by the datastructure, s , and the number of bitprobes used for membership queries, t . Such schemes are often denoted by (n, m, s, t) . To decide membership of an element of \mathcal{U} in the subset \mathcal{S} , the location of a bitprobe might depend on the answers of the previous bitprobes. Such schemes are called *adaptive*. If the location of every bitprobe is independent of the answers we receive in other bitprobes, the schemes are called *non-adaptive*.

Nicholson *et al.* [3] has surveyed the bitprobe model with discussions about current state of the art and a selection of open problems. The reader is advised to refer to their article for further reading.

1.1 The Problem Statements

In this paper, we address the following two problems. The first one is to design an explicit adaptive scheme for storing three elements and deciding membership queries using two bitprobes, i.e. an adaptive scheme for the case when $n = 3$ and $t = 2$. A corollary of this scheme would be a non-adaptive scheme for $n = 3$

and $t = 3$. The second problem we tackle is to design a non-adaptive scheme for storing four elements and deciding membership using four bitprobes, i.e. a non-adaptive scheme for $n = 4$ and $t = 4$.

1.2 Previous Results

Alon and Fiege [1] in their seminal paper presented an adaptive $(n, m, s, 2)$ scheme where $s = o(m)$, which has been further improved by Garg and Radhakrishnan [4], but both of their schemes are non-explicit. The explicit scheme that accommodates the most number of elements for $t = 2$ is due to Radhakrishnan *et al.* [2] where they present a $(2, m, \mathcal{O}(m^{2/3}), 2)$ scheme.

For the case $n = 3$ and $t = 3$ in the non-adaptive bitprobe model, there does not exist any explicit scheme with $s = o(m)$.

In the non-adaptive bitprobe model, the best known explicit scheme with $t = 4$ and $s = o(m)$ is due to Blue [5], and their scheme can handle a subset of size at most three ($n = 3$); the best known non-explicit scheme is due to Alon and Fiege [1], and it can handle $n = o(m)$.

1.3 Our Contribution

We present an explicit adaptive scheme for $n = 3$ and $t = 2$ that uses $s = \mathcal{O}(m^{2/3})$ bits of storage. This scheme improves upon the $(2, m, \mathcal{O}(m^{2/3}), 2)$ scheme of Radhakrishnan *et al.* [2]. It uses a novel technique of mapping the elements of our universe into the integral points of a three dimensional cube, and then looking into the projections of the various points onto the two dimensional faces of the cube. A by-product of this scheme is a non-adaptive scheme for $n = 3$ and $t = 3$, and using $s = \mathcal{O}(m^{2/3})$ bits. Our final scheme is a non-adaptive scheme for $n = 4$ and $t = 4$, and uses $s = \mathcal{O}(m^{2/3})$ bits. This scheme, too, uses the approach described above.

This approach of thinking about the arrangement of elements of our universe as being part of a cube is our central contribution. The versatility of our approach is demonstrated by the fact that it also gives simplified schemes for $n = 2$ and any t (Kesh [6]).

2 Arrangement of Elements

In the three dimensional space with coordinate axes x, y , and z , consider a cube in the first orthant. The cube has sides of magnitude $m^{1/3}$, and it is so placed that one of its vertices lies on the origin, and its sides are parallel to the coordinate axes. The number of points, within and on the cube, with all integer coordinates is m . We place all of the m elements of our universe \mathcal{U} on those points. Going forward, we would refer to an element of \mathcal{U} by the coordinates of the point on which it lies. As an example, an element of \mathcal{U} lying on the point (a, b, c) will be called as the element (a, b, c) . As a consequence, we will use the words ‘element’ and ‘point’ interchangeably.

Based on simple geometric constructions, we now define four distinct partitions of our universe \mathcal{U} . The partitions are named \mathcal{X} , \mathcal{Y} , \mathcal{Z} , and \mathcal{D} . We start by defining the partition \mathcal{X} .

For an element (a, b, c) , the set $X(a, b, c)$ is defined as follows. Draw a line through the point (a, b, c) which is normal to the yz -plane and parallel to the x -axis. The elements of \mathcal{U} that falls on this line belong to the set $X(a, b, c)$. More formally,

$$X(a, b, c) = \{ (d, e, f) \in \mathcal{U} \mid e = b \text{ and } f = c \}.$$

The next two observations would prove that for two elements (a, b, c) and (d, e, f) of our universe, the corresponding sets $X(a, b, c)$ and $X(d, e, f)$ are either equal or disjoint.

Observation 1. *If an element (d, e, f) belongs to the set $X(a, b, c)$, then the sets $X(a, b, c)$ and $X(d, e, f)$ are equal.*

Proof. From the geometric intuition of the sets $X(a, b, c)$ and $X(d, e, f)$, it is clear that the two sets must be equal. There is only one line that is normal to the yz -plane, is parallel to the x -axis, and passes through both the points.

We now argue the same formally. As (d, e, f) belongs to $X(a, b, c)$, we have $e = b$ and $f = c$. So, the point (d, e, f) is actually the point (d, b, c) . Let (g, h, i) be a member of $X(d, b, c)$. Then, we must have $h = b$ and $i = c$. So, the point (g, h, i) is the same as the point (g, b, c) , and hence, it also belongs to $X(a, b, c)$. Therefore, we have $X(d, e, f) \subseteq X(a, b, c)$.

What we have argued so far is the following – if a point (d, e, f) belongs to the set $X(a, b, c)$, then $X(d, e, f) \subseteq X(a, b, c)$. If we can now prove that the point (a, b, c) also belongs to the set $X(d, e, f)$, then we would have established that $X(a, b, c) \subseteq X(d, e, f)$, which in turn would prove the observation.

As it is given that $(d, e, f) \subseteq X(a, b, c)$, so, we have $e = b$ and $f = c$. So, the point (a, b, c) is actually the point (a, e, f) , and from the definition of the set $X(d, e, f)$, it follows that $(a, b, c) = (a, e, f) \in X(d, e, f)$.

Observation 2. *If an element (d, e, f) does not belong to the set $X(a, b, c)$, then the sets $X(a, b, c)$ and $X(d, e, f)$ are disjoint.*

Proof. The geometric interpretation of the sets $X(a, b, c)$ and $X(d, e, f)$ tells us that if the point (d, e, f) does not lie on the line defining the set $X(a, b, c)$, then the line defining the set $X(d, e, f)$ is parallel to the line $X(a, b, c)$. We proceed to argue formally.

As the point (d, e, f) does not lie on the line defining the set $X(a, b, c)$, then either $e \neq b$, or $f \neq c$, or both. Without loss of generality, let us assume that the y -coordinates of the two points are unequal, i.e. $e \neq b$. For an arbitrary point (g, h, i) belonging to the set $X(d, e, f)$, we have $h = e \neq b$. So, the point (g, h, i) cannot belong to the set $X(a, b, c)$. This proves that the sets $X(a, b, c)$ and $X(d, e, f)$ are disjoint.

We now define the partition \mathcal{X} as follows:

$$\mathcal{X} = \left\{ X(0, a, b) \mid 0 \leq a, b < m^{1/3} \right\}.$$

For \mathcal{X} to be a partition, the sets forming \mathcal{X} must be disjoint and they must cover the whole universe \mathcal{U} . Observation 2 guarantees that the first property is satisfied. We prove next that every element of \mathcal{U} belongs to some member of \mathcal{X} . This is easy to see as an element (a, b, c) lies on the line defining the set $X(0, b, c)$.

The following lemma states the size of the partition.

Lemma 1. *The size of the partition \mathcal{X} is $m^{2/3}$.*

Proof. This is an easy consequence of the definition of \mathcal{X} .

The partitions \mathcal{Y} and \mathcal{Z} are similarly defined. We start by defining the sets $Y(a, b, c)$ and $Z(a, b, c)$.

$$\begin{aligned} Y(a, b, c) &= \{ (d, e, f) \in \mathcal{U} \mid d = a \text{ and } f = c \}; \\ Z(a, b, c) &= \{ (d, e, f) \in \mathcal{U} \mid d = a \text{ and } e = b \}. \end{aligned}$$

So, the set $Y(a, b, c)$ is defined by the line through the point (a, b, c) which is normal to the xz -plane and parallel to the y -axis. Similarly, the set $Z(a, b, c)$ is defined by the line through the point (a, b, c) which is normal to the xy -plane and parallel to the z -axis. We can now define the partitions \mathcal{Y} and \mathcal{Z} .

$$\begin{aligned} \mathcal{Y} &= \{ Y(a, 0, b) \mid 0 \leq a, b < m^{1/3} \} \\ \mathcal{Z} &= \{ Z(a, b, 0) \mid 0 \leq a, b < m^{1/3} \} \end{aligned}$$

We can also make the following comment about the size of these partitions.

Lemma 2. *The size of the partitions \mathcal{Y} and \mathcal{Z} are both equal to $m^{2/3}$.*

We will not formally argue any of the facts about the partitions \mathcal{Y} and \mathcal{Z} as the arguments follows closely along the lines of the proof of the properties of partition \mathcal{X} .

The last partition we define is the partition \mathcal{D} . As usual, we start by defining the set $D(a, b, c)$. This set consists of all those points that lie on that line through the point (a, b, c) which lies completely on the xy -plane through the point and has a slope of 45° on that plane. Formally, the set is defined thus.

$$D(a, b, c) = \{ (d, e, f) \in \mathcal{U} \mid f = c \text{ and } d - a = e - b \}$$

Finally, the partition \mathcal{D} is defined as follows.

$$\mathcal{D} = \left\{ D(a, 0, b) \mid 0 \leq a, b < m^{1/3} \right\} \cup \left\{ D(0, a, b) \mid 0 \leq a, b < m^{1/3} \right\}$$

The size of this partition is given in the following lemma.

Lemma 3. *The size of partition \mathcal{D} is $2 \times m^{2/3}$.*

Again, the proof of the properties of this partition follows directly from the geometric interpretation of the sets that form the partition, and we leave it for the reader to argue.

3 The Adaptive Scheme

In this section, we present the first of the two schemes – an explicit scheme in the adaptive bitprobe model for three elements ($n = 3$) and two queries ($t = 2$).

3.1 Our Datastructure

Our datastructure consists of three tables, one for each of the partitions \mathcal{X}, \mathcal{Y} , and \mathcal{Z} of Sect. 2. To abstain from introducing too many notations and risk losing clarity, we abuse the notation for the partitions and use them to denote the tables in our datastructure as well. It will be clear from the context whether the notation \mathcal{X} denotes the partition of \mathcal{U} or the table corresponding to that partition.

For every set in the various partitions, we reserve one bit in the corresponding table in our datastructure. Again, we abuse the notation and use the name for a set to also refer to its corresponding bit in our datastructure. To take an example, $X(a, b, c)$ would refer to a set in partition \mathcal{X} and also the bit reserved for the set in the table \mathcal{X} .

The following lemma follows directly from the Lemmas 1 and 2.

Lemma 4. *The size of our datastructure is $3 \times m^{2/3}$.*

3.2 The Query Scheme

In the adaptive bitprobe model, the query scheme is described by a binary tree, called in the literature as the *decision tree*. This tree tells us the location of a bitprobe, given that the answers of the previous bitprobes are known. The decision tree for our scheme is shown in Fig. 1.

Consider an element (a, b, c) of \mathcal{U} , and we want to determine whether the element belongs to the subset \mathcal{S} . From the definitions of sets and partitions, we know that the element belongs to the set $X(0, b, c)$ in table \mathcal{X} , the set $Y(a, 0, c)$ in \mathcal{Y} , and the set $Z(a, b, 0)$ in table \mathcal{Z} . The decision tree tells us that the first query will be made in table \mathcal{Z} . The location of the bitprobe will be at $Z(a, b, 0)$. If the bit stored in that location is 0, we need to follow the left child and query the location $Y(a, 0, c)$ in table \mathcal{Y} . On the other hand, we need to query the location $X(0, b, c)$ in table \mathcal{X} if the bit stored is 1. We deduce that the element (a, b, c) belongs to the subset \mathcal{S} if and only if the second query returns 1.

3.3 The Storage Scheme

Consider an arbitrary subset $\mathcal{S} = \{ (a_1, b_1, c_1), (a_2, b_2, c_2), (a_3, b_3, c_3) \}$ of our universe \mathcal{U} . The storage scheme describes how to set the bits of our datastructure so that the query scheme can answer correctly. The assignment of bits depend on the how the members of \mathcal{S} are chosen from \mathcal{U} . We describe each such case separately, and provide the proof of correctness alongside it.

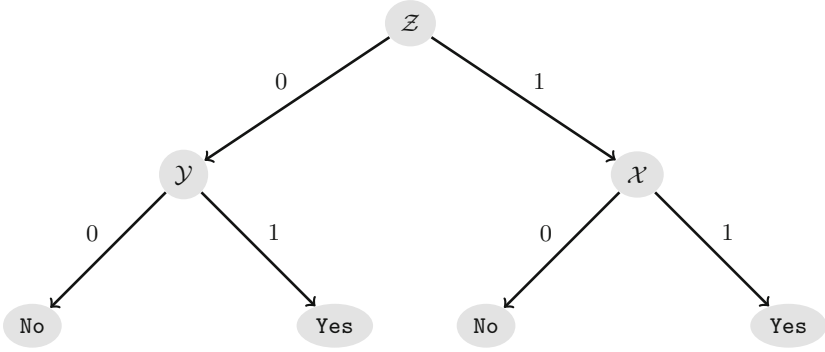


Fig. 1. The decision tree for the adaptive bitprobe scheme.

Case I – Let us consider the scenario when the x -coordinate of the three points are equal, i.e. $a_1 = a_2 = a_3 = a$ (say). The members of \mathcal{S} in this case looks like $\{ (a, b_1, c_1), (a, b_2, c_2), (a, b_3, c_3) \}$.

We set the bits in table \mathcal{Z} as follows – $Z(a, b_1, 0) = Z(a, b_2, 0) = Z(a, b_3, 0) = 1$. The rest of the bits in the table is set to 0. In table \mathcal{Y} , all bits are set to 0. In table \mathcal{X} , we have the following arrangement – $X(0, b_1, c_1) = X(0, b_2, c_2) = X(0, b_3, c_3) = 1$, and the rest of the bits are set to 0.

We now argue that in this case the query algorithm gives correct answers. Consider an element (a', b', c') of \mathcal{U} which upon query in our datastructure got a **Yes**. Then it must got the **Yes** from table \mathcal{X} as all the bits of table \mathcal{Y} are set to 0. To go to table \mathcal{X} , it must get a 1 from table \mathcal{Z} . Without loss of generality, let it get the 1 from the bit $Z(a, b_1, 0)$. So, it must be the case that $a' = a$ and $b' = b_1$, and the element in question is $(a', b', c') = (a, b_1, c')$. In table \mathcal{X} , it will query such a bit whose y -coordinate is b_1 , and which has been set to 1. One such bit is $X(0, b_1, c_1)$. If $b_1 = b_2$, then $X(0, b_2, c_2)$ could also be a possible candidate. If our element queries the bit $X(0, b_1, c_1)$, then $c' = c_1$. So, the element in question is $(a', b', c') = (a, b_1, c_1)$, which indeed is a member of \mathcal{S} . We can argue similarly in other cases as well.

Case II – Let us now consider the scenario where two elements of \mathcal{S} have the same x -coordinate, and the remaining element's x -coordinate is different from the other two. Without loss of generality, let $a_1 = a_2 = a$ (say), and $a \neq a_3$. So, the elements of the set \mathcal{S} are $\{ (a, b_1, c_1), (a, b_2, c_2), (a_3, b_3, c_3) \}$. We would consider two subcases, one in which the y -coordinate of the third element is equal to the y -coordinate of one of the first two elements, the other in which the y -coordinate of the third element is distinct from the y -coordinates of the first two elements.

Case II(A) – We consider the first subcase here, in which the y -coordinate of the third element is equal to the y -coordinate of one of the first two elements. Without loss of generality, let $b_1 = b_3 = b$ (say). So, the elements of \mathcal{S} are

$\{ (a, b, c_1), (a, b_2, c_2), (a_3, b, c_3) \}$. A further complication might arise if $b = b_2$. We consider the possibility in the subcases below.

Case II(A)(a) – Let us consider the scenario where $b \neq b_2$. So, the elements of \mathcal{S} remain as $\{ (a, b, c_1), (a, b_2, c_2), (a_3, b, c_3) \}$. If such is the case, in table \mathcal{Z} , we set $Z(a, b', 0) = 1$, for all $b' \neq b$. We also set $Z(a_3, b, 0) = 1$. The rest of the bits of table \mathcal{Z} are set to 0. In table \mathcal{Y} , we set $Y(a, 0, c_1) = 1$, and all the other bits to 0. Finally, in table \mathcal{X} , we set $X(0, b_2, c_2) = X(0, b, c_3) = 1$, and the other bits to 0.

We now argue that this arrangement of bits in our datastructure is correct. Consider an element (a', b', c') that got a **Yes** upon query in our datastructure. Since both tables \mathcal{Y} and \mathcal{X} have one or more bits set to 1, (a', b', c') could have got its **Yes** answer from either of them.

Let us first consider the case where the element (a', b', c') went to table \mathcal{Y} and got its 1 from there. $Y(a, 0, c_1)$ is the only bit in that table that is set to 1. So, we have $a' = a$ and $c' = c_1$, which makes the element (a', b', c') to be (a, b', c_1) . The only bit in table \mathcal{Z} which is set to 0 and has its x -coordinate as a is $Z(a, b, 0)$, and our element must query this bit. So, we further have $b' = b$. Thus the element $(a', b', c') = (a, b, c_1)$, a member of \mathcal{S} .

The other case to consider is when the element (a', b', c') went to table \mathcal{X} and got a **Yes**. The way bits are set in \mathcal{X} , we have either $b' = b_2, c' = c_2$ or $b' = b, c' = c_3$. So, the element could be one of (a', b_2, c_2) and (a', b, c_3) . The only way (a', b_2, c_2) can get a 1 in table \mathcal{Z} is by querying the bit $Z(a, b_2, 0)$, which implies that $a' = a$, and $(a', b', c') = (a, b_2, c_2)$. (a', b, c_3) can get a 1 from table \mathcal{Z} by querying the bit $Z(a_3, b, 0)$. So, we have $a' = a_3$, and hence $(a', b', c') = (a_3, b, c_3)$. So, in all of the cases, (a', b', c') turns out to be a member of \mathcal{S} .

Case II(A)(b) – We next consider the case where $b = b_2$. The elements of \mathcal{S} , now, would be $\{ (a, b, c_1), (a, b, c_2), (a_3, b, c_3) \}$. This case is not too dissimilar from the previous case in the arrangement of its elements. The assignment of table \mathcal{Z} remains unchanged. In table \mathcal{Y} , we set $Y(a, 0, c_1) = Y(a, 0, c_2) = 1$, and all the other bits to 0. Finally, in table \mathcal{X} , we set $X(0, b, c_3) = 1$, and the other bits to 0.

The proof of correctness is similar to the previous case, and we omit it for the sake of brevity.

Case II(B) – We now consider the second subcase where the y -coordinate of the third element is distinct from the y -coordinate of the other two elements. So, the set \mathcal{S} is $\{ (a, b_1, c_1), (a, b_2, c_2), (a_3, b_3, c_3) \}$. If such is the case, we set $Z(a, b_1, 0) = Z(a, b_2, 0) = Z(a_3, b_3, 0) = 1$, and the rest of the bits of table \mathcal{Z} to 0. All bits of table \mathcal{Y} are set to 0. The table \mathcal{X} , only the following bits are set to 1 – $X(0, b_1, c_1), X(0, b_2, c_2), X(0, b_3, c_3)$.

We again argue that only elements of \mathcal{S} upon query in our datastructure will get a **Yes** answer. Let (a', b', c') be an arbitrary element which got a **Yes** from our datastructure. Then, it must go to table \mathcal{X} to get that answer, as all the elements of table \mathcal{Y} are set to 0. Let it be the case that it got a 1 by querying the bit $X(0, b_1, c_1)$. So, it must be the case that $b' = b_1$ and $c' = c_1$, and hence $(a', b', c') = (a', b_1, c_1)$. For such an element to get a 1 from table \mathcal{Z} , it must

query $Z(a, b_1, 0)$, and thus $a' = a$. So, we have $(a', b', c') = (a, b_1, c_1)$, a member of \mathcal{S} . We can similarly argue that if (a', b', c') queries some other bit in \mathcal{X} to get a 1, it will still be a member of \mathcal{S} .

Case III – The last case to consider is when the x -coordinates of all the members of \mathcal{S} are distinct. We set the bits of the three tables as follows. In table \mathcal{Z} , $Z(a_1, b_1, 0) = Z(a_2, b_2, 0) = Z(a_3, b_3, 0) = 0$, and the rest of the bits are set to 1. In table \mathcal{Y} , $Y(a_1, 0, c_1) = Y(a_2, 0, c_2) = Y(a_3, 0, c_3) = 1$, and the other bits are set to 0. All bits in table \mathcal{X} are set to 0.

Once again we argue that if an element, say (a', b', c') , upon query in our datastructure got a **Yes**, then it must be a member of \mathcal{S} . The element can get a **Yes** only from table \mathcal{Y} . Without loss of generality, let us assume that it queried $Y(a_1, 0, c_1)$. In this case, we have $a' = a_1, c' = c_1$, and hence $(a', b', c') = (a_1, b', c_1)$. In table \mathcal{Z} , it must get a 0 so that it can go to table \mathcal{Y} , and the only bit which is set to 0 and whose x -component is a_1 is the bit $Z(a_1, b_1, 0)$. This gives us $b' = b_1$, and thus $(a', b', c') = (a_1, b_1, c_1)$, which is a member of \mathcal{S} .

This concludes the description of the storage scheme and our proof of correctness. We can now summarise the conclusions of this section as follows.

Theorem 5. *There is an explicit adaptive $(3, m, 3 \times m^{2/3}, 2)$ scheme.*

This scheme also gives rise to a non-adaptive scheme. If we decide to probe all the tables of our decision tree irrespective of the findings on our first query in table \mathcal{Z} , then we would have made three bitprobes in our datastructure, instead of two. More importantly, the scheme now becomes non-adaptive as the location of every query is fixed. The query scheme, on getting the results of the three queries, can now decide membership by consulting the decision tree of Fig. 1. Thus, we can claim the following.

Corollary 6. *There is an explicit non-adaptive $(3, m, 3 \times m^{2/3}, 3)$ scheme.*

4 The Non-adaptive Scheme

We present our final scheme, an explicit non-adaptive scheme for four elements ($n = 4$) and four queries ($t = 4$).

4.1 Our Datastructure

Our datastructure consists of four tables, one for each partition of Sect. 2, namely \mathcal{X} , \mathcal{Y} , \mathcal{Z} , and \mathcal{D} . As in the previous section, we refrain from introducing too many notations and use the notations for the partitions to denote the tables in our datastructure as well. Furthermore, we reserve one bit for every set in a partition in its corresponding table. As before, we use the same name for the sets in the partitions and the corresponding bits in the tables. As an example, $D(a, b, c)$ would refer to a set in partition \mathcal{D} and also the bit reserved for the set in the table \mathcal{D} .

The following lemma follows directly from the Lemmas 1, 2, and 3.

Lemma 7. *The size of our datastructure is $5 \times m^{2/3}$.*

4.2 The Query Scheme

If we want to ascertain the membership in set \mathcal{S} of an element (a, b, c) of \mathcal{U} , we query its corresponding bits in each of the tables of our datastructure, namely the bit $X(0, b, c,)$ in table \mathcal{X} , the bit $Y(a, 0, c,)$ in table \mathcal{Y} , and so on. Upon receiving the query answers, we apply the majority function to determine the membership in \mathcal{S} . If the majority of the bits returned is 1, then and only then we declare that the element in question is a member of \mathcal{S} .

4.3 The Storage Scheme

In this section, we describe how to set the bits of our datastructure such that the query scheme can correctly answer membership queries. The way bits are set depends upon how the members of \mathcal{S} are chosen. We discuss below each such case, and provide proof of correctness of the scheme alongside it.

In the following discussion, we assume that $\mathcal{S} = \{(a_1, b_1, c_1), (a_2, b_2, c_2), (a_3, b_3, c_3), (a_4, b_4, c_4)\}$.

Case I – Let us assume that all the four members of \mathcal{S} lie in the same xy -plane, i.e. $c_1 = c_2 = c_3 = c_4 = c$ (say). In this case, we set the bits corresponding to each member of \mathcal{S} in tables \mathcal{X}, \mathcal{Y} , and \mathcal{Z} to 1. The rest of the bits in all of the tables, including \mathcal{D} , are set to 0. So, for the element (a_1, b_1, c) , $X(0, b_1, c) = Y(a_1, 0, c) = Z(a_1, b_1, 0) = 1$, and similarly for the other members of \mathcal{S} .

We now provide the correctness proof of our scheme in this scenario. Let us assume that an element (a', b', c') , upon query in our datastructure, has got the majority of its answers as 1. As all of the bits of table \mathcal{D} is set to 0, it must get 1 from each of the tables \mathcal{X}, \mathcal{Y} , and \mathcal{Z} . As only four bits in table \mathcal{Z} are set to 1, let us assume that it got its 1 from the bit $Z(a_1, b_1, 0)$. This implies that $a' = a_1$ and $b' = b_1$, and hence the point in question is (a_1, b_1, c') . This point also got a 1 when it queried table \mathcal{X} . This table also has four bits set to 1 corresponding to the four members of \mathcal{S} . The element (a_1, b_1, c') will query such of bit of \mathcal{X} which is set to 1 and whose y -coordinate is b_1 . $X(a_1, b_1, c)$ is one such bit. If $b_1 = b_2$, then $X(a_2, b_2, c)$ is also a possible candidate. Let us assume that (a_1, b_1, c') queried the set $X(a_1, b_1, c)$. It immediately gives us $c' = c$, and thus the point (a', b', c') is actually the point (a_1, b_1, c) , which indeed is a member of \mathcal{S} . We can similarly argue if other sets are queried.

Case II – We now assume that three members of \mathcal{S} are in one xy -plane, and the other member is in a different plane. Let the three members in the same plane be the first three members of \mathcal{S} . So, we have $c_1 = c_2 = c_3 = c$ (say), and $c_4 \neq c$. This scenario gives rise to two different arrangement of the elements that have to be handled differently.

Case II(A) – Let us assume that in partition \mathcal{Z} , the element (a_4, b_4, c_4) lies in one of the sets of the other three elements of \mathcal{S} . Without loss of generality, let that element be (a_1, b_1, c) . From Observation 1, if $(a_4, b_4, c_4) \in Z(a_1, b_1, c)$ then $Z(a_4, b_4, c_4) = Z(a_1, b_1, c) = Z(a_1, b_1, 0)$. Thus, we have $a_1 = a_4$ and $b_1 = b_4$. So, the four elements of \mathcal{S} are $\{(a_1, b_1, c), (a_2, b_2, c), (a_3, b_3, c), (a_1, b_1, c_4)\}$. If that is

the case, we set the bits corresponding to each member of \mathcal{S} in tables \mathcal{X} , \mathcal{Y} , and \mathcal{Z} to 1. The rest of the bits in all of the tables, including \mathcal{D} , are set to 0.

If an element (a', b', c') got a majority of its query answers as 1, it must get those from tables \mathcal{X} , \mathcal{Y} , and \mathcal{Z} . In table \mathcal{Z} , it can either query the bit $Z(a_1, b_1, 0)$ or any of the other two sets that are set to 1. If it queries $Z(a_1, b_1, 0)$, we have $a' = a_1$ and $b' = b_1$. In table \mathcal{X} , it will query such a set whose y -coordinate is b_1 and which is set to 1. Two such sets are $X(0, b_1, c)$ and $X(0, b_1, c_4)$. If $b_1 = b_2$, then $X(0, b_2, c_2)$ is another candidate. If our element queries $X(0, b_1, c_1)$, then we have $c' = c$, and thus $(a', b', c') = (a_1, b_1, c)$ which is a member of \mathcal{S} . We can similarly argue the other cases.

If instead of querying the set $Z(a_1, b_1, 0)$ in table \mathcal{Z} , if (a', b', c') queries any of the other sets that are set to 1, say $Z(a_2, b_2, 0)$, we can similarly argue that the element (a', b', c') will again be a member of \mathcal{S} . We leave the details in such cases to the reader.

Case II(B) – We now consider the scenario when the element (a_4, b_4, c_4) does not lie in any of the sets of the other three elements in partition \mathcal{Z} . Then for the element (a_4, b_4, c_4) , we set the bits $X(0, b_4, c_4)$ and $Y(a_4, 0, c_4)$ in tables \mathcal{X} and \mathcal{Y} , respectively, to 1. Also in table \mathcal{D} , we set one of the bits $D(a_4 - b_4, 0, c_4)$ or $D(0, b_4 - a_4, c_4)$ to 1, according as a_4 is greater than or less than b_4 . Without loss of generality, let us assume that $a_4 \geq b_4$. For the other three elements, we set their bits in tables \mathcal{X} , \mathcal{Y} , and \mathcal{Z} to 1. All the other bits in all of the tables are set to 0.

Consider an element (a', b', c') which got majority of its queries as 1. The z -coordinate of the point could either be equal to c , or be equal to c_4 , or it could be distinct from both c and c_4 . We consider each of these cases separately.

Case II(B)(i) – Let c' be different from c and c_4 . Then, its queries into the tables \mathcal{X} and \mathcal{Y} must return 0. This is due to the fact that the z -coordinates of all the bits that are set to 1 in tables \mathcal{X} and \mathcal{Y} are either c or c_4 . To take an example, if it queried the bit $X(0, b_1, c)$ and hence got a 1, then $c' = c$, which contradicts our assumption. So, in this scenario, the element (a', b', c') cannot get more than two 1s, and hence no such element can get a **Yes** answer.

Case II(B)(ii) – Let c' be equal to c_4 . To get a majority of its answer as 1, it must get a 1 from one of the tables \mathcal{X} and \mathcal{Y} . Let it be the table \mathcal{X} . We want such a set of \mathcal{X} which has its z -coordinate equal to c_4 and is set to 1. The only set satisfying the constraints is $X(0, b_4, c_4)$, which gives us $b' = b_4$.

We now look at query it made in table \mathcal{D} . If it returned a 0, then for the sake of majority, its query into table \mathcal{Y} must return 1. The set whose z -coordinate is c_4 and is set to 1 in this table is $Y(a_4, 0, c_4)$, and hence $a' = a_4$. So, we have $(a', b', c') = (a_4, b_4, c_4)$, a member of \mathcal{S} . If the query made in table \mathcal{D} returned a 1, then it must be the case that $a' - (a_4 - b_4) = b' - 0 = b_4 - 0$, which implies that $a' = a_4$. So, we have $(a', b', c') = (a_4, b_4, c_4)$, a member of \mathcal{S} . Other cases similarly follows.

Case II(B)(iii) – The final case we look into is when $c' = c$. As all bits that are set to 1 in table \mathcal{D} has the z -coordinate equal to c_4 , the element (a', b', c) must

get a 0 upon query in this table. So, the query answers from all the other tables must be 1. In table \mathcal{Z} , there are three sets whose corresponding bits are set to 1. If the element queries the set $Z(a_1, b_1, 0)$, then we have $a' = a_1$ and $b' = b_1$. So, the element (a', b', c') must be (a_1, b_1, c) , a member of \mathcal{S} . We can argue the other cases similarly.

Case III – We now consider the scenario when at most two members of \mathcal{S} belong to the same xy -plane. This case is much simpler than the previous ones – we set the bits corresponding to the members of \mathcal{S} in tables \mathcal{X} , \mathcal{Y} and \mathcal{D} to 1, and the rest of the bits, including those of table \mathcal{Z} to 0.

The proof in this case is also similar to the proofs done in the previous cases. The only thing that we have to consider is when an xy -plane contains two elements, and when all the elements are on a separate xy -plane. As a demonstration we prove one such case next, and leave the rest of the cases to the reader.

Let the first two elements of \mathcal{S} , namely (a_1, b_1, c_1) and (a_2, b_2, c_2) , belong to the same xy -plane. We also assume that $a_1 \geq b_1$ and $a_2 \geq b_2$. This implies in the bits $D(a_1 - b_1, 0, c_1)$ and $D(a_2 - b_2, 0, c_2)$ are set to 1. Consider the element (a', b', c') which also belongs to this plane. We prove that in this scenario, if the element (a', b', c') upon query in our datastructure got a **Yes**, then it must be a member of \mathcal{S} . From our assumptions, we have $c_1 = c_2 = c' = c$ (say). As all the bits of table \mathcal{Z} has its bits set to 0, (a', b', c) must get 1 from rest of the tables.

In table \mathcal{X} , the only bits set to 1 and with z -coordinate equal to c is $X(0, b_1, c)$ and $X(0, b_2, c)$. Let the element query the set $X(0, b_1, c)$. This gives us $b' = b_1$, and hence the element in question is (a', b_1, c) . The bits of table \mathcal{Y} that are set to 1 and have the z -coordinate c are $Y(a_1, 0, c)$ and $Y(a_2, 0, c)$, and our element must have queried one of these sets. So, the element (a', b', c') could be one of (a_1, b_1, c) and (a_2, b_1, c) . If the element is (a_1, b_1, c) , it is already a member of \mathcal{S} and we have nothing to prove.

We now consider the case of $(a', b', c') = (a_2, b_1, c)$. The bits of table \mathcal{D} that are set to 1 with z -coordinate c are $D(a_1 - b_1, 0, c)$ and $D(a_2 - b_2, 0, c)$. This tells us that if the element (a', b', c') is actually the element (a_1, b_2, c) , then either $a_1 = a_2$ or $b_1 = b_2$. In both of these cases, the element is a member of \mathcal{S} .

We now conclude the description of our storage scheme and the proof of correctness. The following theorem summarises the result of this section.

Theorem 8. *There is an explicit non-adaptive $(4, m, 5 \times m^{2/3}, 4)$ scheme.*

5 Conclusion

We have proposed a novel technique for visualising the arrangement of elements in the bitprobe model, that of arranging the elements on the integral points of a suitably sized three dimensional cube. This gives us, what essentially is, three new results, one in the domain of adaptive bitprobe model, and two in the non-adaptive model. The technique can be extended to higher dimensional cubes, and has already given interesting results in the two query adaptive bitprobe model (Kesh [6]). We believe that this technique will give improved results in several other scenarios in both the adaptive and non-adaptive model.

References

1. Alon, N., Feige, U.: On the power of two, three and four probes. In: Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, 4–6 January 2009, pp. 346–354 (2009)
2. Radhakrishnan, J., Raman, V., Srinivasa Rao, S.: Explicit deterministic constructions for membership in the bitprobe model. In: auf der Heide, F.M. (ed.) ESA 2001. LNCS, vol. 2161, pp. 290–299. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44676-1_24
3. Nicholson, P.K., Raman, V., Srinivasa Rao, S.: A survey of data structures in the bitprobe model. In: Brodник, A., López-Ortiz, A., Raman, V., Viola, A. (eds.) Space-Efficient Data Structures, Streams, and Algorithms: Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday. LNCS, vol. 8066, pp. 303–318. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40273-9_19
4. Garg, M., Radhakrishnan, J.: Set membership with a few bit probes. In: Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, 4–6 January 2015, pp. 776–784 (2015)
5. Blue, R.: The bit probe model for membership queries: non-adaptive bit queries. Master of Science Thesis (2009)
6. Kesh, D.: On adaptive bitprobe schemes for storing two elements. In: Gao, X., Du, H., Han, M. (eds.) COCOA 2017. LNCS, vol. 10627, pp. 471–479. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-71150-8_39

Faster Network Algorithms Based on Graph Decomposition

Manas Jyoti Kashyop¹, Tsunehiko Nagayama², and Kunihiro Sadakane²(✉)

¹ Department of Computer Science and Engineering,
Indian Institute of Technology Madras, Chennai, India
`manasjk@cse.iitm.ac.in`

² Department of Mathematical Informatics, Graduate School of Information Science
and Technology, The University of Tokyo, Tokyo, Japan
`sada@mist.i.u-tokyo.ac.jp`

Abstract. We propose faster algorithms for the maximum flow problem and related problems based on graph decomposition. Our algorithms first construct indices (data structures) from a given graph, then use them for solving the problems. A basic problem is an all pairs maximum flow problem, which consists of two stages. In a preprocessing stage we construct an index, and in a query stage we process the query using the index. We can solve all pairs maximum flow problem and minimum cut problem using the indices. Time complexities of our algorithms depend on the size of the maximum triconnected component in the graph, say r . Our algorithms run faster than known algorithms if r is small. The maximum flow problem can be solved in $\mathcal{O}(nr)$ time, which is faster than the best known $\mathcal{O}(nm)$ algorithm [Orlin 2013] if $r = o(m)$, where n and m are the numbers of vertices and edges, respectively.

1 Introduction

In this paper, we propose faster algorithms for the maximum flow problem and other related problems based on graph decomposition. The basic problem we consider is the following:

Max Flow Indexing Problem(MFIP): Given a directed graph, we preprocess it to construct an index. Then given two vertices s, t , we compute the value of the maximum $s - t$ flow using the index.

Let n and m be the number of vertices and edges of a given graph (network), respectively, throughout this paper.

We also consider the following problems using the algorithm for MFIP:

Maximum $s - t$ flow problem: Given a directed graph and two vertices s, t , compute the maximum $s - t$ flow.

The work was supported in part by JSPS KAKENHI 16K12393.

All Pairs Max Flow Problem (APMFP): Given a directed graph, compute the values of the maximum flow between every pair of vertices.

Minimum Cut Problem (MCP): Given a directed graph, compute the value of the minimum cut of the graph.

The MFIP is a problem consisting of two stages: a preprocessing stage for constructing an index from a given graph, and a query stage for computing the desired value using the index given two vertices. If the graph is static and we need to compute the maximum flow values for many pairs of vertices, by using an index (data structure) constructed in the preprocessing stage, the queries can be done faster than computing the value without preprocessing. The extreme case is that in the preprocessing stage we compute maximum flow values for every pair of vertices and store them in a two-dimensional array. Then a query is trivially done in constant time. However this approach is not efficient because the index uses $\mathcal{O}(n^2)$ space even if the input size is linear in n , and a naive algorithm for constructing the index solves the maximum flow problem $\mathcal{O}(n^2)$ times. Another extreme case is to use the input graph as the index. Then the preprocessing time is constant but the query time is equal to that for solving a maximum flow problem. Therefore there is a trade-off between preprocessing time, query time, and index size. For a problem consisting of a preprocessing stage and a query stage, an algorithm is called a $\langle p(n), q(n) \rangle$ time algorithm if the preprocessing time is $p(n)$ and the query time is $q(n)$.

1.1 Related Work

The maximum flow problem is well studied [7, 8, 10, 18, 20]. Among them, the fastest algorithm runs in $\mathcal{O}(nm)$ time. There are also algorithms for special cases of graphs, for example the $\mathcal{O}(n \log \log n)$ time algorithm for undirected planar graphs [17], the $\mathcal{O}(n \log n)$ time algorithm for directed planar graphs [4], the linear time algorithm for constant tree-width graphs [13].

For MFIP and APMFP on undirected graphs, the Gomory-Hu tree [11] can be used as an index. However it is known [3] that there is no such structure for directed graphs. For constant tree-width graphs, APMFP is solved in $\mathcal{O}(n^2 + \gamma^3 \log \gamma)$ time on planar graphs, or $\mathcal{O}(n^2 + \gamma^4 \log \gamma)$ time if $m = \mathcal{O}(n)$ [2], where γ is the number of hammocks obtained by the hammock decomposition [9].

For the minimum cut problem, there are $\mathcal{O}(nm + n^2 \log n)$ time algorithm for undirected graphs [19] and $\mathcal{O}(nm \log(n^2/m))$ time algorithm for general graphs [14]. Tables 1 and 2 summarize complexities of existing algorithms and our algorithms.

1.2 Our Contribution

We propose faster algorithms for the above problems based on graph decomposition. Namely, we use BC-trees [16] and SPQR-trees [23] for decomposition. A BC-tree represents the biconnected components of a graph, and an SPQR-tree represents the triconnected components of a biconnected graph. The performance

Table 1. Complexities of max-flow problem, APMF, and MCP where n, m, γ, r denote the number of nodes, the number of edges, the number of hammocks, and the maximum size of triconnected components, respectively.

Problem	Reference	Graph class	Time complexity
Maximum flow	[18, 20]	General	$\mathcal{O}(nm)$
	[17]	Undirected planar	$\mathcal{O}(n \log \log n)$
	[4]	Directed planar	$\mathcal{O}(n \log n)$
	[13]	Constant tree-width	$\mathcal{O}(n)$
	Theorem 4	General	$\mathcal{O}(m + nr)$
APMFP	[2]	Constant tree-width	$\mathcal{O}(n^2)$
	[2]	Planar	$\mathcal{O}(n^2 + \gamma^3 \log \gamma)$
	[2]	$m = \mathcal{O}(n)$	$\mathcal{O}(n^2 + \gamma^4 \log \gamma)$
	Theorem 7	General	$\mathcal{O}(m + nr^3 + n^2)$
MCP	[19]	Undirected	$\mathcal{O}(nm + n^2 \log n)$
	[14]	General	$\mathcal{O}(nm \log(n^2/m))$
	Corollary 1	General	$\mathcal{O}(m + nr^3 + n^2)$

Table 2. Complexities of MFIP. If $T_1(k, n) = \lambda(k, n)$, $T_2(k, n) = 1$. If $T_1(k, n) = 1$, $T_2(k, n) = \alpha(n)$. The functions $\lambda(k, n)$ and $\alpha(n)$ are the inverse Ackermann function defined in Sect. 2.4

Reference	Graph class	Complexity
[2]	Constant tree-width	$\langle \mathcal{O}(nT_1(k, n)), \mathcal{O}(T_2(k, n)) \rangle$
Theorem 5	General	$\langle \mathcal{O}(m + nT_1(k, n) + nr^3), \mathcal{O}(T_2(k, n)) \rangle$

of our algorithms depends on a parameter of graphs: the size of the maximum tri-connected components, denoted by r . If a given graph is decomposed into small triconnected components, our algorithms run faster than existing algorithms. For example, for MCP, our algorithm is faster than [19] if $r = \mathcal{O}(n^{1/3})$, and faster than [14] if $r = \mathcal{O}(m^{1/3})$. For the maximum flow problem, our algorithm is faster than [20] if $r = o(m)$. For MFIP, the algorithm of Arikati et al. [2] works efficiently for constant tree-width graphs. However the time complexities are doubly exponential to the tree-width, and finding the tree decomposition is NP-hard. On the other hand, the time complexity of our algorithm is polynomial in r . We also provide support for dynamic update and queries for $s - t$ max flow in time $\mathcal{O}(nr)$ which is better than recomputing the best known algorithm if $r = o(m)$.

2 Preliminaries

2.1 BC-Trees

BC-trees [16] are trees representing the biconnected component decomposition of a connected graph, defined as follows.

Definition 1. Let $G = (V, E)$ be a connected graph.

- (i) A vertex $v \in V$ is called a cut vertex of G if removing v from G makes the graph disconnected.
- (ii) A maximal connected subgraph of G that does not have any cut vertex is called a block of G .
- (iii) A tree $T = (B \cup C, F)$ is called a BC-tree of G if it satisfies the following.
 - C is the set of cut vertices of G and B is the set of blocks of G .
 - Any $c \in C$ and any $b \in B$ are adjacent in T i.e. $(b, c) \in F \iff$ the block corresponding to b contains the cut vertex c .

2.2 SPQR Trees

SPQR tree [23] data structure is used to maintain triconnected components of a biconnected graph.

Following updates are supported by dynamic SPQR trees:

InsertEdge(v_1, v_2): Insert an edge between vertices v_1 and v_2 .

InsertVertex(v, v_1, v_2): Split the edge (v_1, v_2) into two edges (v_1, v) and (v, v_2) by inserting vertex v .

SPQR tree supports the following query operation:

ThreePaths(v_1, v_2): Returns TRUE if three vertex disjoint paths exist between vertices v_1 and v_2 .

The following results are known.

Theorem 1 [23]. Let G be a biconnected graph with n vertices and m edges. An SPQR tree data structure uses $\mathcal{O}(n)$ space and supports operation ThreePaths in $\mathcal{O}(1)$ time and can be constructed in $\mathcal{O}(n + m)$ time.

Theorem 2 [23]. An SPQR tree data structure for biconnected graphs supports a sequence of k operations consisting of ThreePaths, InsertEdge or InsertVertex in time $\mathcal{O}(k\alpha(k, n))$ where n is the number of InsertVertex operations and $\alpha(k, n)$ denotes the Ackermann’s function inverse.

2.3 Mimicking Networks

In this section, we review mimicking networks [13].

Let $N = (G = (V, E), c)$ be a network and let $Q = \{q_1, \dots, q_k\} \subseteq V$. If a function $f : E \rightarrow \mathbf{R}_{\geq 0}$ satisfies $f(e) \leq c(e)$ for all $e \in E$, and $i_f(v) = 0$ for all $v \in V \setminus Q$, where $i_f(v) = \sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e)$, $\delta^+(v)$ denotes the set of edges going out of v and $\delta^-(v)$ denotes the set of edges entering v and

f is called a Q -flow. For a Q -flow f , $(i_f(q_1), \dots, i_f(q_k))$ is called the *external flow* w.r.t. f . If we consider all feasible Q -flows, the set of all external flows define a subset of \mathbf{R}^Q . We call it the external flow pattern of N w.r.t. Q . It is proved [13] that any external flow pattern is expressed by a set of $2^k + 1$ linear inequalities. External flow patterns can be also expressed by mimicking networks. Let $N' = (G' = (V', E'), c')$ be a network satisfying $Q \subseteq V'$. If the external flow pattern of N' w.r.t. Q coincides with that of N w.r.t. Q , N' is called a mimicking network of N with terminal set Q . Hagerup et al. [13] proved the following.

Lemma 1 [13]. *For any network and its vertex subset Q , there exists a mimicking network $N' = (G' = (V', E'), c')$ with terminal set $Q \subseteq V'$ such that $|V'| \leq 2^{2^{|Q|}}$.*

Therefore, if the number of terminals is constant, the size of the mimicking network is also constant. Furthermore, for undirected graphs with four terminals, there exists a mimicking network with five nodes [5].

2.4 Tree Product Queries

We use algorithms for the tree product query problem, defined as follows.

Tree product query. Given a semi-group (S, \circ) , a tree $T = (V, E)$, and a function $f : V \rightarrow S$, compute $f(u) \circ f(w_1) \circ f(w_2) \circ \dots \circ f(v)$ for given $u, v \in V$, where (u, w_1, \dots, v) denotes the $u - v$ path.

If the preprocessing time is $p(n)$ and the time for a query is $q(n)$, we denote the time complexity by $\langle p(n), q(n) \rangle$. The following is known.

Theorem 3 [1, 6]. *There exists algorithms for tree product queries with time complexity $\langle O(kn\lambda(k, n)), O(k) \rangle$ and $\langle O(kn), O(\alpha(n)) \rangle$ for any $k > 0$ where $\lambda(k, n)$ and $\alpha(n)$ are the inverse Ackermann functions. The index sizes are $O(kn\lambda(k, n))$ and $O(kn)$, respectively.*

3 Preprocessing

In this section, we show preprocessing algorithms for solving max-flow problem using SPQR trees.

3.1 Constructing D_0 Data Structure

First we give a data structure D_0 which stores in each node μ of the SPQR tree for a graph $G = (V, E)$, the edge capacity of the mimicking network corresponding to the node μ .

If μ is an S-node, its skeleton is a polygon, consisting of the reference edge $\{u, v\}$ and a path connecting u and v . Here the reference edge can be considered

as the network outside of the skeleton. If we see the skeleton from outside, it is the path between u and v . Then we can regard the path as an edge between u and v . Its capacity is the minimum among edges on the path if the graph is undirected. If the graph is undirected, we create two directed edges (u, v) and (v, u) . Their edge capacities are defined analogously.

If μ is a P-node, its skeleton is a graph with two vertices u, v and k multiple edges between them. Among k edges, one is the reference edge. Therefore we store an edge between u and v whose capacity is the summation of those of the edges except the reference edge.

If μ is an R-node, its skeleton is a triconnected graph. Let $\{u, v\}$ be the reference edge. If the graph is undirected, we compute the minimum cut value c between u and v in the skeleton without the edge $\{u, v\}$, and we store an edge with capacity c . If the graph is directed, we compute both $u - v$ and $v - u$ minimum cut values and store two edges whose capacities are those values.

We analyze the time complexity of the above algorithm. For S- and P-node, it takes time proportional to the number of edges in the SPQR tree, which is $\mathcal{O}(m)$. For each R-node, we compute max-flow constant times. Let n_i and r_i be the numbers of nodes and edges in the skeleton of a node μ_i of the SPQR tree. Then it takes $\mathcal{O}(n_i r_i)$ time for computing max-flow [20]. Therefore the total time for computing the D_0 data structure is

$$\sum_{\mu_i} \mathcal{O}(n_i r_i) = \mathcal{O}\left(\sum_{\mu_i} n_i r_i\right) = \mathcal{O}(nr)$$

where $r = \max r_i$ is the maximum size of skeletons.

Lemma 2. *Given an SPQR tree of a biconnected graph which has n nodes and the maximum size of whose triconnected components is r , the D_0 data structure is stored in $\mathcal{O}(m)$ space and constructed in $\mathcal{O}(nr)$ time.*

3.2 Constructing D_1 Data Structure

Next we compute D_1 data structure which stores in each node of the SPQR tree, a mimicking network with four terminals, that is, of constant size. Let μ be a node of the SPQR tree, v_1, v_2, \dots, v_{k-1} be its children, and $\{u, v\}$ be the reference edge of μ .

If μ is an S-node, let $q_1 = u, q_2, \dots, q_k = v$ be the nodes of the skeleton. The capacities of edges $\{q_1, q_2\}, \{q_2, q_3\}, \dots, \{q_{k-1}, q_k\}$ are stored in the D_0 data structure. For node v_i , we store the following graph with at most four terminals $\{q_i, q_{i+1}, u, v\}$. The graph has at most two edges: $\{u, q_i\}, \{q_{i+1}, v\}$. The edge capacities are the minimum of those of $\{q_1, q_2\}, \dots, \{q_{i-1}, q_i\}$, the minimum of those of $\{q_i, q_{i+1}\}, \dots, \{q_{k-1}, q_k\}$, respectively. That is, the graph is obtained by deleting the edge $\{q_{i-1}, q_i\}$ merging other edges into two. The edge capacities are computed in $\mathcal{O}(k)$ time as follows. If we know the minimum edge capacity of $\{q_1, q_2\}, \dots, \{q_{j-1}, q_j\}$, the minimum edge capacity after adding another edge $\{q_j, q_{j+1}\}$ is computed in constant time. By repeating this, we obtain all the edge capacities.

If μ is a P-node, let e_1, e_2, \dots, e_k be the edges of the skeleton, and e_k be the reference edge. Assume that the edge e_i corresponds to the children v_i . Then for each v_i , we store a graph with two terminals $\{u, v\}$. The edge capacity is $\sum_{j=1}^{k-1} c(e_j) - c(e_i)$ where $c(e_j)$ is the capacity of e_j for undirected graphs. For directed graphs it is computed analogously. We can compute those graphs for all children of μ in $\mathcal{O}(k)$ time.

If μ is an R-node, for each child v_i of μ , we compute a mimicking network $M(G_{\text{sk}}[\mu] \setminus \{\{u, v\}, \{s, t\}\}, \{u, v, s, t\})$ where s, t are end points of the reference edge of v_i . If the skeleton of μ has n_i nodes and r_i edges, the mimicking network is computed in $\mathcal{O}(n_i r_i)$ time. Therefore for each R-node, it takes $\mathcal{O}(n_i r_i^2)$ time. The total time for all R-nodes is $\mathcal{O}(nr^2)$.

Lemma 3. *Given an SPQR tree of a biconnected graph which has n nodes and the maximum size of whose triconnected components is r , the D_1 data structure is stored in $\mathcal{O}(m)$ space and constructed in $\mathcal{O}(nr^2)$ time.*

3.3 Constructing D_2 Data Structure

The D_2 data structure is to store for each pair $\{s_1, t_1\}, \{s_2, t_2\}$ of edges of each R-node μ whose reference edge is $\{u, v\}$, the mimicking network with at most six terminals $M(G_{\text{sk}}[\mu] \setminus \{\{u, v\}, \{s_1, t_1\}, \{s_2, t_2\}\}, \{u, v, s_1, t_1, s_2, t_2\})$. For the node μ with n_i nodes and r_i edges, it takes $\mathcal{O}(n_i r_i^3)$ time. Then the total time is $\mathcal{O}(nr^3)$. The space is $\mathcal{O}(mr)$ because for each node we store at most $r - 1$ mimicking networks of constant size.

Lemma 4. *Given an SPQR tree of a biconnected graph which has n nodes and the maximum size of whose triconnected components is r , the D_2 data structure is stored in $\mathcal{O}(mr)$ space and constructed in $\mathcal{O}(nr^3)$ time.*

4 Computing $s - t$ Max Flow in $\mathcal{O}(m + nr)$ Time

In this section, we show an algorithm for computing $s - t$ max flow in $\mathcal{O}(m + nr)$ time using the D_0 data structure. First we consider an input graph is biconnected. We compute the SPQR tree in $\mathcal{O}(n + m)$ time, then construct the D_0 data structure in $\mathcal{O}(nr)$ time.

From each of the given nodes s, t , we choose an arbitrary Q-node containing the node. Let μ_s, μ_t be the nodes, and p be their lowest common ancestor. Let $v_0 = \mu_s, v_1, \dots, v_d = p$ be the nodes in the SPQR tree on the path from μ_s to p . For each node v_i in v_1, \dots, v_{d-1} , we compute the mimicking network with at most four terminals by merging the mimicking networks for siblings of v_i . This is actually the same as $M(G_{\text{sk}}[v_{i+1}] \setminus \{\{u, v\}, \{x, y\}\}, \{u, v, x, y\})$, which is in the D_1 data structure, where $\{u, v\}$ and $\{x, y\}$ are the reference edges of v_{i+1} and v_i , respectively. Note that we do not compute those networks for all children of an R-node; only for the child having the Q-node for s in its subtree. Similarly for nodes between μ_t and p , we compute mimicking networks. We also compute for

nodes between p and the root of the SPQR tree, the mimicking networks with at most four terminals.

Finally we merge all the mimicking networks computed above. Because two mimicking networks adjacent in the tree have two common vertices, we can merge them. Let n_1, r_1 and n_2, r_2 be the number of nodes and edges in the two skeletons, respectively. The time complexity to merge the mimicking networks is $\mathcal{O}((n_1 + n_2)(r_1 + r_2)) = \mathcal{O}((n_1 + n_2)r)$. Then the total time complexity is $\mathcal{O}(nr)$. Now we have a mimicking network with four terminals s, s', t, t' where s' and t' are the other end points of the edges in the Q-node containing s and t . By adding the edges $\{s, s'\}$ and $\{t, t'\}$ to the mimicking network and computing the $s - t$ minimum cut, we obtain the answer. This is done in constant time because the mimicking network is of constant size.

Once the value of the $s - t$ max flow is obtained, we can compute the flow itself. If the external flow of a mimicking network is fixed, we can obtain the flow in a skeleton by computing max flows constant times. And once the flow value of an edge of a skeleton is fixed, we can recursively compute the flow for the skeleton. The time complexity is the same as computing the max flow value.

Next we consider a general graph. First we compute the BC-tree in $\mathcal{O}(n + m)$ time. If s and t belong to the same biconnected component, we are done. Otherwise, for all blocks in the BC-tree on the path from the one containing s to the one containing t , we compute minimum cut values, and obtain the result. The time complexity is $\mathcal{O}(m + nr)$.

Theorem 4. *For a graph with n nodes and m edges whose maximum triconnected component is of size r , an $s - t$ max flow is computed in $\mathcal{O}(m + nr)$ time.*

5 Algorithms for MFIP

In this section we give algorithms for the MFIP. The results are summarized as follows.

Theorem 5. *For a directed network with n vertices and m edges,*

- (i) *after $\mathcal{O}(m + n\lambda(k, n) + nr^3)$ -time preprocess, using an index of size $\mathcal{O}(m + n\lambda(k, n) + mr)$, the value of the maximum flow is computed in constant time, or*
- (ii) *after $\mathcal{O}(m + nr^3)$ -time preprocess, using an index of size $\mathcal{O}(m + mr)$, the value of the maximum flow is computed in $\mathcal{O}(\alpha(n))$ time,*
- (iii) *after $\mathcal{O}(m + nr^2)$ -time preprocess, using an index of size $\mathcal{O}(m)$, the value of the maximum flow is computed in $\mathcal{O}(\alpha(n) + r^2)$ time,*

where r is the maximum size of triconnected components in the underlying undirected graph.

The proof is in the following subsections.

5.1 Algorithms for Fast Queries

To solve MFIP in a biconnected graph, we use the D_1 and the D_2 data structures and the tree product query data structure. In each node of the SPQR tree, a mimicking network is stored as D_1 . Because merging of mimicking networks is associative, we can use the data structure for tree product queries for those mimicking networks. The preprocess and query times are either $\mathcal{O}(n\lambda(k, n))$ and $\mathcal{O}(1)$, or $\mathcal{O}(n)$ and $\mathcal{O}(\lambda(k, n))$ for any $k \geq 0$.

Assume that vertices s, t are given as a query. For each of s, t , we choose an arbitrary Q-node containing the node. Let μ_s, μ_t be the nodes, and p be their lowest common ancestor. Let q_s and q_t be children of p on the path between μ_s and p and on the path between μ_t and p , respectively. Then the mimicking network M_s between μ_s and q_s is computed by using the tree product query data structure. Similarly the mimicking network M_t between μ_t and q_t is computed. We also compute the mimicking network M_p for nodes between p and the root of the SPQR tree using the tree product query data structure. Then we merge M_s, M_t, M_p , and the mimicking network $M(G_{\text{sk}}[p] \setminus \{\{u, v\}, \{s_1, t_1\}, \{s_2, t_2\}\}, \{u, v, s_1, t_1, s_2, t_2\})$ where $\{u, v\}$ are the common vertices between M_p and $M(G_{\text{sk}}[p])$, $\{s_1, t_1\}$ are the common vertices between M_s and $M(G_{\text{sk}}[p])$, and $\{s_2, t_2\}$ are the common vertices between M_t and $M(G_{\text{sk}}[p])$. Because $M(G_{\text{sk}}[p])$ is stored in the D_2 data structure and it is of constant size (six terminals), we can merge them in constant time, and compute the max flow value in constant time.

If the graph is not biconnected, in the preprocessing stage we construct the BC-tree and for each biconnected component we construct the SPQR tree and the tree product query data structure. Then for the BC-tree, we preprocess it for tree product queries.

5.2 An Algorithm with Small Index

Here we give an algorithm using $\mathcal{O}(m)$ space based on the D_1 data structure and the tree product query data structure. The algorithm is different from that in the previous subsection that we do not use the D_2 data structure. Therefore for a query we have to solve max-flow problems in two nodes (p and the root). Because the number of edges in a skeleton is at most r , the max-flow can be solved in $\mathcal{O}(r^2)$ time.

6 Dynamic Update and Query

Our algorithm supports the following dynamic update operations:

- **InsertVertex**(v, v_1, v_2): Split the edge (v_1, v_2) into two edges (v_1, v) and (v, v_2) by inserting vertex v . If c was the capacity for the edge (v_1, v_2) then capacity for each of the edge (v_1, v) and (v, v_2) is c .
- **InsertEdge**(v_1, v_2, c): Add an edge between vertices v_1 and v_2 with capacity c .
- **UpdateCapacity**(v_1, v_2, c): Change the capacity of the edge (v_1, v_2) to c .

Whenever an edge is deleted from graph, we will updating the capacity of the edge to 0 without actually deleting the edge. We use the same procedures as mentioned in [23] to update the SPQR tree. We update D_1 data structures for every update. D_0 data structures are also updated in similar way and within the same update time.

InsertVertex(v, v_1, v_2):

After the update, edges (v_1, v) and (v, v_2) have the same capacity as that of the edge (v_1, v_2) . Therefore updating D_1 data structures will take $O(1)$ time.

InsertEdge(v_1, v_2, c):

We will divide the update operation into two phases: local update and global update. If D_1 data structure for a child of an R-node or a P-node or an S-node α gets updated, we need to update D_1 data structures for all the other children of α . But we will update the data structures only in the path starting from the new Q-node till the root of the SPQR tree. We delay the update for all other children of α until those structures are required in a Query. We mark those children of α as dirty using a dirty bit for every child node. Therefore if α has k children then marking all of them as dirty will take $O(k)$ time. Let λ be the new Q-node corresponding to edge (v_1, v_2) . Let r be the maximum size of triconnected components of the given graph. Suppose v_1 and v_2 have exactly one common allocation node and let it be μ . Suppose μ is an R-node. Let p be the parent of node μ . To update the SPQR tree, an edge between λ and μ is added.

Local update: D_1 data structure for λ is computed in constant time. Updating D_1 data structure for μ will take time $O(MF(G_{sk}(\mu)))$. We mark all the other children of node μ as dirty. Remaining updates of D_1 data structure will be handled in *global* update. So time required for local updates is $O(|V(G_{sk}(\mu))| \cdot |E(G_{sk}(\mu))|) + O(|E(G_{sk}(\mu))|) = O(nr)$. Where n is currently the number of vertices in the graph G .

Global update: We update D_1 structures from p till root of the SPQR tree. Let (i, j) be an edge in that path. We have already updated D_1 data structure for node i and let k be the parent of j . Suppose j is an S-node or P-node. In that case number of children of j is $O(n)$. For each children updating D_1 structure takes constant time and hence updating for all the children will take $O(n)$ time. If j is a R-node then updating D_1 data structure for j will take time $O(G_{sk}(j))$. We will update the D_1 structure for only one child which is in the path from new Q-node to j . We will mark all other children of j as dirty. Now since maximum number of R-nodes in the path till root node is $O(n)$, total time for updating D_1 data structure along the path will take $O(n.r)$ time. Let T' be the total time required for marking the children of all the R-nodes. Then, $T' = O(|E(G_{sk}(r_1))|) + O(|E(G_{sk}(r_2))|) + \dots + O(|E(G_{sk}(r_k))|)$. Since skeletons of the R-nodes are edge disjoint, $T' = O(m)$, where m is currently the number of edges in the entire graph G . No other D_1 data structure need to be updated because it is not difficult to observe that subnetwork corresponding to other branches of the SPQR tree are edge disjoint.

If μ is an S-node then local update takes $\mathcal{O}(1)$ time. If μ is a P-node then local update takes $\mathcal{O}(n)$ time. Suppose v_1 and v_2 does not have any common allocation node. After updating the SPQR tree, local update step takes $\mathcal{O}(nr)$ time. In all these cases, global update step takes $\mathcal{O}(nr)$ time.

UpdateCapacity(v_1, v_2, c):

After this update SPQR tree remains the same. D_1 structure is updated starting from the Q-node where capacity has been changed till the root of the SPQR tree. Time complexity analysis for this update is similar to global update and hence time required is $\mathcal{O}(nr)$.

Query for $s - t$ Max Flow:

As explained in Sect. 4, computation of $s - t$ max flow involves D_0 and D_1 data structure. After selecting arbitrary Q-nodes containing s and t , finding a dirty ancestor if any will take $\mathcal{O}(n)$ time. If there is one, then we will update the D_1 data structure along the path till root node. A similar analysis as seen in *global* update will give us Query time as $\mathcal{O}(nr)$.

Theorem 6. *In dynamic setting, operation `InsertVertex` can be supported in $\mathcal{O}(1)$ time and `InsertEdge`, `UpdateCapacity` and `Query for $s - t$ max flow` can be supported in $\mathcal{O}(nr)$ time. Here r is currently the maximum size of triconnected components in the underlying undirected graph.*

7 Algorithms for Other Problems

The APMFP can be solved by computing the values of maximum flows for every pair of vertices using the D_2 data structure and the tree product query data structure.

Theorem 7. *The values of maximum flows of all pairs of vertices is computed in $\mathcal{O}(m + nr^3 + n^2)$ time.*

The minimum cut problem is also solved trivially by finding the maximum of all maximum flow values.

Corollary 1. *The value of the minimum cut is computed in $\mathcal{O}(m + nr^3 + n^2)$ time.*

8 Concluding Remarks

We have proposed faster algorithms for network problems, especially the maximum flow and the minimum cut problems, based on graph decomposition. Different from an existing work [2] based on the tree decomposition whose time complexity is doubly exponential to the tree-width, time complexities of our algorithms depend polynomially on a parameter r , the size of the maximum triconnected component. More importantly, triconnected component decomposition can be done in linear time, whereas finding a tree decomposition with

minimum tree-width is NP-hard if the tree-width is not constant. Though $r = m$ in the worst case, our algorithms are faster than existing ones for small r cases. For the $s - t$ maximum flow problem, our algorithm runs in $\mathcal{O}(nr)$ time, which is faster than the fastest algorithm [20] if $r = o(m)$.

Our approach based on triconnected component decomposition can be easily applied for other network problems such as the shortest-path problem, network reliability problem. Our future work is to extend the scope of our approach and to show practical performance on real networks.

References

1. Alon, N., Schieber, B.: Optimal preprocessing for answering on-line product queries. Technical report, Tel-Aviv University (1987)
2. Arikati, S.R., Chaudhuri, S., Zaroliagis, C.D.: All-pairs min-cut in sparse networks. *J. Algorithms* **29**, 82–110 (1998)
3. Benczúr, A.A.: Counterexamples for directed and node capacitated cut-trees. *SIAM J. Comput.* **24**, 505–510 (1995)
4. Borradaile, G., Klein, P.: An $O(n \log n)$ algorithm for maximum st -flow in a directed planar graph. In: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm, pp. 524–533 (2006)
5. Chaudhuri, S., Subrahmanyam, K.V., Wagner, F., Zaroliagis, C.D.: Computing mimicking networks. *Algorithmica* **20**, 31–49 (2000)
6. Chazelle, B.: Computing on a free tree via complexity-preserving mappings. *Algorithmica* **2**, 337–361 (1987)
7. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM* **19**, 248–264 (1972)
8. Ford, L.R., Fulkerson, D.R.: Maximal flow through a network. *Can. J. Math.* **8**, 399–404 (1956)
9. Frederickson, G.N.: Using cellular graph embeddings in solving all pairs shortest paths problems. In: Proceedings of 30th Annual Symposium on Foundations of Computer Science, pp. 448–453 (1989)
10. Goldberg, A.V., Tarjan, R.E.: A new approach to the maximum-flow problem. *J. ACM* **35**, 921–940 (1988)
11. Gomory, R.E., Hu, T.C.: Multi-terminal network flows. *J. Soc. Ind. Appl. Math.* **9**, 551–570 (1961)
12. Gutwenger, C., Mutzel, P.: A linear time implementation of SPQR-trees. In: Marks, J. (ed.) GD 2000. LNCS, vol. 1984, pp. 77–90. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44541-2_8
13. Hagerup, T., Katajainen, J., Nishimura, N., Radge, P.: Characterizations of k -terminal flow networks and computing network flows in partial k -trees. In: Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 641–649 (1995)
14. Hao, J.X., Orlin, J.B.: A faster algorithm for finding the minimum cut in a directed graph. *J. Algorithms* **17**, 424–446 (1994)
15. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.* **13**, 338–355 (1984)
16. Hopcroft, J., Tarjan, R.: Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM* **16**, 372–378 (1973)

17. Italiano, G.F., Nussbaum, Y., Sankowski, P., Wulff-Nilsen, C.: Improved algorithms for min cut and max flow in undirected planar graphs. In: Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing, pp. 313–322 (2011)
18. King, V., Rao, S., Tarjan, R.E.: A faster deterministic maximum flow algorithm. *J. Algorithms* **17**, 447–474 (1994)
19. Nagamochi, H., Ibaraki, T.: Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM J. Discrete Math.* **5**, 54–66 (1988)
20. Orlin, J.B.: Max flows in $O(nm)$ time, or better. In: Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing, pp. 765–774 (2013)
21. Robertson, N., Seymour, P.D.: Graph minors. III. Planar tree-width. *J. Comb. Theory Ser. B* **36**, 49–64 (1984)
22. Shing, M.T., Hu, T.C.: A decomposition algorithm for multi-terminal networks flows. *Discrete Appl. Math.* **13**, 165–181 (1986)
23. Di Battista, G., Tamassia, R.: On-line maintenance of triconnected components with SPQR-trees. *Algorithmica* **15**(4), 302–318 (1996)
24. Niedermeier, R.: Invitation to fixed-parameter algorithms. Oxford Lecture Series in Mathematics and its Application (2006). ISBN 9780198566076

An Improvement of the Algorithm of Hertli for the Unique 3SAT Problem

Tong Qin^(✉)  and Osamu Watanabe 

School of Computing, Tokyo Institute of Technology, Tokyo, Japan
{qin5,watanabe}@is.titech.ac.jp

Abstract. We propose a simple idea for improving the randomized algorithm of Hertli for the Unique 3SAT problem. Using recently developed techniques, we can derive from this algorithm the currently the fastest randomized algorithm for the general 3SAT problem.

1 Introduction

The 3SAT problem is a problem of deciding whether a given 3CNF formula is satisfiable, where a *3CNF formula* is a propositional Boolean formula expressed as a conjunction of 3-clauses consisting of at most three literals. (A *k-clause* is a disjunction of k literals, and a *literal* is either a Boolean variable or a negated Boolean variable). The Unique 3SAT problem that we discuss in this paper is a variation of the 3SAT problem where we may assume that a given input formula has at most one satisfying assignment. The 3SAT problem is one of the typical NP-complete problems, and in particular, it has been a target of obtaining better exponential-time algorithms. For a given 3CNF formula F over n variables, the straightforward approach for solving the problem is to check for every possible assignment to n variables whether it satisfies F , i.e., F is evaluated true by the assignment. This needs $\tilde{O}(2^n)$ -time¹ in the worst case. While it has been believed that no polynomial-time algorithm exists for the 3SAT problem, we can expect an algorithm that has a better exponential-time bound. In fact, researchers have proposed various clever algorithms for the 3SAT problem that have better exponential-time bounds.

We review briefly some of important algorithms for the 3SAT problem. Note that such algorithms are usually defined for more general k SAT problems for any $k \geq 3$; but here we focus only on the 3SAT problem. In 1997, Paturi et al. [5] proposed a randomized algorithm (which is now called PPZ) that runs in $\tilde{O}(1.588^n)$ -time. In 1998, Paturi et al. [4] improved it and obtain a faster algorithm (which is now called PPSZ). They showed that it runs in $\tilde{O}(1.364^n)$ -time for the 3SAT problem; though they also showed that it runs in $\tilde{O}(1.308^n)$ -time

¹ Throughout this paper, we use n , the number of variables of an input formula, as a size parameter. Following the standard convention on this topic, we ignore the polynomial factor for discussing the time complexity of algorithms, and by $\tilde{O}(T(n))$ we denote $O(T(n)p(n))$ for some polynomial p .

for the Unique 3SAT problem, it was left open to show that this time bound holds for the 3SAT problem in general. Soon after, Schönning [6] proposed a randomized algorithm of a different type that runs in $\tilde{O}(1.334^n)$ -time for the 3SAT problem. Since then several improvements have been reported until Hertli [1] proved that the $\tilde{O}(1.308^n)$ -time bound of PPSZ indeed holds for the 3SAT problem in general. More recently, there are some more improvements in relation to the Unique 3SAT problem. Though an improvement is extremely small (an improvement of, say, the 25th digit of the exponential base $1.308 \dots$), Hertli [2] showed a way to improve the Unique 3SAT problem, which we refer as Hertli’s algorithm in this paper. Furthermore, Hertli [3] (Theorem 6.2) and Scheder and Steinberger [7] gave general methods to make use of a randomized algorithm for the Unique 3SAT problem for solving the general 3SAT problem while keeping similar exponential-time bounds. Thus, based on Hertli’s algorithm, we can show an algorithm for the 3SAT problem that is better than PPSZ (though the improvement is extremely small).

In this paper we give a simple idea for improving Hertli’s algorithm and show that it indeed gives a better exponential-time bound than Hertli’s algorithm (Theorem 2). Therefore (again the improvement is extremely small) we can apply the methods of Hertli and Scheder and Steinberger to derive an algorithm that has a yet better time bound over the currently known algorithms.

The improvement over PPSZ that Hertli’s algorithm achieves is obtained by considering several cases and by giving a better treatment for each case. One of the key ideas is to use (together with PPSZ) Wahlström’s algorithm [8] that performs better than PPSZ if a target formula consists of small number of clauses, which is guaranteed that the degree of the formula is bounded. The *degree_k* of a variable x of a 3CNF formula F is the number of k -clauses of F containing x or \bar{x} as a literal. We say (in this paper) that a formula is *b-degree_k bounded* if the largest degree_k of its variables is at most b . For any CNF formula F and any set W of variables of F , let $F \setminus W$ denote a formula obtained by removing all clauses containing some variable in W . In order to use Wahlström’s algorithm, Hertli introduced the following condition separating the “dense/sparse” cases determined by a parameter Δ , $0 < \Delta < 1$:

A formula F is Δ -*sparse* if there exists a set W of at most Δn variables of F such that $F \setminus W$ is 4-degree₃ bounded. Otherwise, F is Δ -*dense*.

For the sparse case (that is, the case where we assume that a target formula F is Δ -sparse) Hertli’s algorithm executes the sparse-case algorithm. This algorithm guesses the above set W of variables in a straightforward way and then use the combination of PPSZ and Wahlström’s algorithm on $F \setminus W$, improving the efficiency of PPSZ. On the other hand, the dense-case algorithm is executed for the case where we assume that a target formula F is Δ -dense. Starting from $F_3 := F$, it first repeats $\Delta n/2$ iterations of collecting a clause that is chosen randomly from clauses of F_3 that contain a randomly chosen degree₃ ≥ 5 variable (whose existence is guaranteed by the Δ -denseness), while modifying F_3 by removing clauses containing the variables of the selected clause. Let F_2 be

the set of $\Delta n/2$ clauses obtained by this process. Then there is a way to assign values to the variables in F_2 that is better than the random guess. By using this way of choosing a partial assignment, the dense-case algorithm can search a sat. assignment for F more efficiently than PPSZ.

We notice that the above set W of variables can be found when creating F_2 in the dense-case algorithm. That is, while the dense-case algorithm tries to get clauses from F_3 , if it cannot find any $\text{degree}_3 \geq 5$ variable in F_3 , then a set of variables removed from F_3 in the dense-case algorithm is indeed the set W witnessing the Δ -sparseness of F in the above condition. Thus, the sparse-case algorithm also begin with the iteration of the dense-case algorithm for computing W . In this way, we can avoid the straightforward guess part of the sparse-case algorithm. In order to justify this idea, we introduce a “soft” condition replacing the above “hard” condition for the Δ -sparseness/-denseness.

In this paper, we assume that the reader is familiar with [2], and we will skip reviewing several technical details common with [2].

Preliminaries

We prepare some of the key notions for our discussion. For any $k \geq 2$, a k CNF formula is a CNF formula consisting of k -clauses having exactly k literals. On the other hand, ($\leq k$)CNF formula consists of ($\leq k$)-clauses, clauses having at most k literals. A 1-clause is often called a *unit clause*.

Consider any CNF formula F over n Boolean variables. We use $\text{vbl}(F)$ to denote the set of variables of F . We use x to denote a variable of F , and for a variable x , its literal, i.e., x or its negation \bar{x} , is denoted by $\ell(x)$. Throughout this paper we use F to denote a current target CNF formula (usually, a given input to an algorithm that we discuss) and use V and n to denote respectively $\text{vbl}(F)$ and $|\text{vbl}(F)|$, that is, the set of Boolean variables of F and its size.

An *assignment* is a mapping α from V to $\{0, 1\}$. We sometimes consider a *partial assignment* whose value is undefined on some variable(s). For any assignment α and any subset U of V , we use $\alpha|_U$ to denote the partial assignment that is the same as α on U and that is undefined on $V \setminus U$. For any (partial) assignment, we use $F[\alpha]$ to denote a formula obtained by assigning a value $\alpha(x)$ (if it is defined) to each variable x and then simplifying the resulting formula. In general, by, e.g., $F[x_1 \leftarrow 0, x_2 \leftarrow 1]$ we mean a formula obtained by simplifying F after assigning these values to its variables x_1 and x_2 .

Consider any formula F with n variables. A *satisfying assignment* (in short, sat. assignment) of F is an assignment such that $F[\alpha] = 1$, i.e., true. We say that α is a unique sat. assignment (of F) if it is a sat. assignment and there is no other assignment satisfying F . For any $d \geq 1$, d -isolated sat. assignment (of F) is a sat. assignment that has no other sat. assignment within Hamming distance d . It is easy to see that a unique sat. assignment is maximally isolated, that is, n -isolated, and hence, d -isolated for any $d \leq n$. For any sat. assignment α (of F), a clause of F is called *critical* (for a variable x w.r.t. α) if only $\ell(x)$ in the clause is evaluated 1 under the assignment α . For any $d \geq 1$, if α is a d -isolated, then every variable of F has at least one critical clause. We say that a partial assignment β (for F) is *consistent* with another (partial) assignment

α (for F) if for each variable x of F , either $\beta(x) = \text{undefined}$ or $\beta(x) = \alpha(x)$ holds. If F has a unique sat. assignment α , then for any partial assignment β consistent with α , α remains the unique assignment of $F[\beta]$.

Algorithms we consider in this paper are all randomized algorithms unless explicitly stated otherwise. In general, for any algorithm described as a procedure A and any input instance w , by $A(w)$ we mean the execution of A on the input w , which is sometimes regarded as a random process determined by the random choices of A . Throughout the following technical discussion, by a *sat. algorithm* we mean a procedure that yields one of the sat. assignments of a given satisfiable formula (or reports “failure” and stops if a sat. assignment is not obtained). For any sat. algorithm A , its *success probability* is a function mapping the size parameter n to the smallest probability that the algorithm yields a sat. assignment for any satisfiable formula (that also satisfies a certain assumption defined in each context) with n variables. In this paper, we propose sat. algorithms with subexponential-time bounds and have success probabilities better than the one for PPSZ that is $2^{-(S+o(1))n}$ where S is a constant $0.386 \dots$ (see Lemma 2 of the next section). Typically, we consider a procedure X with time complexity bounded above by $2^{o(n)}$ and success probability (on a certain subset of satisfiable 3CNF formulas) bounded below by $2^{-(S-\varepsilon+o(1))n}$ for some constant $\varepsilon > 0$. In this paper, we call the amount ε the *efficiency improvement* of X (on the target instance set). Clearly, the inverse of the success probability gives an expected number of executions of X to get a sat. assignment, and we can easily define a bounded error randomized algorithm for the decision problem with the corresponding exponential-time bound.

2 Hertli’s Algorithm

We recall Hertli’s algorithm and some of the facts from [2] necessary for our discussion. We follow [2] and use the same algorithms and lemmas including the usage of symbols as much as possible (except for correcting some minor errors and introducing additional notation).

For main sat. algorithms, Hertli’s algorithm uses PPSZ [4] and Wahlström’s algorithm [8]. First consider PPSZ and discuss two minor changes on PPSZ introduced in [2] for simplifying analysis. We start with some notions.

Definition 1. *For any s , we say that a CNF formula F s -implies a literal ℓ if there exists a subformula $G \subseteq F$ with $|G| \leq s$ such that all satisfying assignment of G set ℓ to 1. (Throughout this paper, we use $s_0(n) = \log n$ for s . This choice is enough to guarantee the performance of PPSZ as stated below).*

Definition 2. *For any CNF formula F over n variables and for any of its variable x , we say that x is forced (during the execution of PPSZ) if F $s_0(n)$ -implies its literal $\ell(x)$. Otherwise, we say that x is guessed (during the execution of PPSZ).*

Definition 3. A random placement π is a mapping from V to $[0, 1]$ such that for each x , $\pi(x)$ is chosen independently and uniformly at random from $[0, 1]$. In general, for any parameter $p \in [0, 1)$, a random ($\geq p$)-placement π is a mapping from V to $[p, 1]$ defined in the same way.

With these notions, we formally define a procedure PPSZ stated as Algorithm 1. This PPSZ is different from the original PPSZ in the following two points: (i) the s_0 -implication is used to “force” an assignment of a variable instead of applying the s_0 -bounded resolution; and (ii) a random placement is used instead of a random permutation. It is shown [2] that the important properties of the original PPSZ are kept under these modifications, which are stated as lemmas below (Lemmas 1 and 2).

In the following, as a lower bound of the success probability of a sat. algorithm A, we consider the probability that A yields some particular target sat. assignment. For simplifying our statement we use, unless otherwise stated explicitly, F to denote any satisfiable (≤ 3)CNF-formula and α_* to denote any of its sat. assignment α_* regarded as a target assignment. We may assume that F also satisfies a certain condition given in each context. Let E_A denote the event that $A(F)$ yields α_* . Below let $\text{PPSZ}(F|\pi_0, \alpha_*)$ denote the execution of PPSZ on F by using π_0 for its random placement π and α_* for β . Note that PPSZ is deterministic if we fix π and β in the algorithm.

Lemma 1. Consider any satisfiable (≤ 3)CNF F . For any placement π , define $G(\pi)$ to denote the number of guessed variables during the execution $\text{PPSZ}(F|\pi, \alpha_*)$. Then we have $\Pr[E_{\text{PPSZ}}] \geq \text{Exp}_\pi[2^{-G(\pi)}] \geq 2^{\text{Exp}_\pi[-G(\pi)]}$.

Lemma 2. Define S and S_p by

$$S = \int_0^1 \left(1 - \min \left\{ 1, \frac{r^2}{(1-r)^2} \right\} \right) dr \quad \text{and} \quad S_p = \int_p^1 \left(1 - \min \left\{ 1, \frac{r^2}{(1-r)^2} \right\} \right) dr$$

Consider any satisfiable (≤ 3)CNF F that has a $\log s_0(n)$ -isolated sat. assignment α_* . For any π , let $G(\pi)$ be (as the above lemma) the number of guessed variables during the execution $\text{PPSZ}(F|\pi, \alpha_*)$. Also, for any $p \in [0, 1/2]$, let $G_p(\pi)$ be the number of variables with placement $> p$ that are guessed during the execution $\text{PPSZ}(F|\pi, \alpha_*)$. Then we have

- (1) $\text{Exp}_\pi[G(\pi)] = (S + o(1))n$,
- (2) for any $p \in [0, 1/2]$, $\text{Exp}_\pi[G_p(\pi)] = (S_p + o(1))n$,
- (3) $S = 2 \ln 2 - 1 = 0.386 \dots$, and $S_p = S - p + I(p)$, where $I(p) := \int_0^p \frac{r^2}{(1-r)^2} dr$.

From these lemmas, we have the following bounds.

Lemma 3. Let F be any satisfiable (≤ 3)CNF that has a $\log s_0(n)$ -isolated sat. assignment α_* . Then $\Pr[E_{\text{PPSZ}}]$ is at least $2^{-(S+o(1))n}$. Furthermore, suppose that we pick every variable of F with prob. p independently, and let V_p be the resulting set. Let E_{PPSZ, V_p} denote the event that $\text{PPSZ}(F[\alpha_*|_{V_p}])$ returns $\alpha_*|_{V \setminus V_p}$. Then we have $\text{Exp}_{V_p}[\log \Pr[E_{\text{PPSZ}, V_p}]] \geq \text{Exp}_\pi[-G_p(\pi)] = -(S_p + o(1))n$.

Algorithm 1. PPSZ **input:** a (≤ 3) CNF formula F

```

1:  $V \leftarrow \text{vbl}(F)$ ;  $n \leftarrow |V|$ 
2: Choose  $\beta$  u.a.r. from all assignments on  $V$ 
3: Choose  $\pi$  u.a.r. as a random placement of  $V$ 
4: Let  $\alpha$  be a partial assignment on  $V$ , initially undefined for all  $x \in V$ ,
5: for  $x \in V$ , in ascending order of  $\pi(x)$  do
6:   if  $F$   $s_0(n)$ -implies  $\ell(x)$  then set  $\alpha(x)$  to satisfy this literal ( $\leftarrow x$  is forced)
7:   else  $\alpha(x) \leftarrow \beta(x)$  ( $\leftarrow x$  is guessed)
8:    $F \leftarrow F[x \rightarrow \alpha(x)]$ 
9: end for
10: return  $\alpha$  if  $\alpha$  is a sat. assignment (otherwise, report “failure”)

```

We consider the above bound $2^{-(S+o(1))n}$ as a target, and we propose algorithms for certain types of input formulas with some “efficiency improvements” that is, algorithms that have success probabilities larger than $2^{-(S-\varepsilon+o(1))n}$ for some $\varepsilon > 0$. The first such example is PPSZ itself; PPSZ performs better if a given formula has many variables with more than one critical clauses [2].

Lemma 4. *Let F be any (≤ 3) CNF formula that has a $\log s_0(n)$ -isolated sat. assignment α_* . If Δn variables of F have more than one critical clause, then $\Pr[\text{E}_{\text{PPSZ}}] \geq 2^{-(S-0.00145 \cdots \Delta + o(1))n}$. Furthermore, if F has more than Δn variables that have a critical 2-clause, then $\Pr[\text{E}_{\text{PPSZ}}] \geq 2^{-(S-0.0353\Delta + o(1))n}$.*

Wahlström [8] proposed a deterministic algorithm for solving the CNF-SAT problem. Here we denote by WAHLSTROM the following procedure based on Wahlström’s algorithm.

Lemma 5. *For any CNF formula F with no unit clause that has average degree at most d , $4 < d \leq 5$, WAHLSTROM(F) computes one of its sat. assignment in time $\tilde{O}(2^{0.115707 \cdots (d-1)n})$.*

Note that the time complexity of WAHLSTROM is not $2^{o(n)}$. Thus, we consider the following randomized procedure W_rand: For a given input CNF formula F over n variables with average degree $\leq d$, W_rand(d, F) executes the above procedure with probability $2^{-0.115707(d-1)n}$, and otherwise it stops the computation immediately with failure. Clearly, the success probability of W_rand(F) is $2^{-0.115707(d-1)n}$, and its expected running time is $2^{o(n)}$.

Now we consider Hertli’s algorithm HERTLI stated as Algorithm 2. First we remark on our way to state algorithms by pseudo codes. In the following algorithms such as Algorithm 2, we consider several cases on a given formula, and execute a sat. algorithm on the formula or its subformula that works efficiently for each case. Though it is not stated explicitly, by, e.g., “Execute PPSZ(F')” we mean to (i) execute the procedure PPSZ on F' , (ii) compute a sat. assignment of the input formula of the procedure based on the obtained sat. assignment, and then (iii) terminate the computation by yielding the computed sat. assignment. Clearly, if the execution at the step (i) fails, then the computation is terminated

reporting “failure.” Note also that it may not be easy to determine which case actually holds for a given formula. Therefore, we consider all the cases *in parallel*. By “**assume Φ then \dots** ” in our algorithm descriptions, we mean to execute the “ \dots ” part in parallel with the other cases assuming that Φ holds.

Algorithm 2. HERTLI

input: a (≤ 3)CNF formula F , **parameter:** $\Delta_1, \Delta_2, \delta_3, \delta_4, p$

- 1: $V \leftarrow \text{vbl}(F)$; $n \leftarrow |V|$;
 - 2: **assume** F has more than $\Delta_1 n$ var.s with more than one critical clause **then**
 - 3: Execute PPSZ(F)
 - 4: **assume** otherwise **then**
 - 5: Choose u.a.r. a size $\Delta_1 n$ subset W_1 of V and an assignment α_1 on W_1 ;
 (assume below that W_1 and α_1 are correctly chosen)
 - 6: $F' \leftarrow F[\alpha_1]$; $V' \leftarrow \text{vbl}(F')$; $n' \leftarrow |V'|$
 - 7: **assume** F' is Δ_2 -dense **then** (what follows is the dense-case algorithm)
 - 8: $F_2 \leftarrow \text{GetInd2Clauses}(F')$
 - 9: Execute DensePPSZ $_p(F', F_2)$
 - 10: **assume** otherwise **then** (what follows is the sparse-case algorithm)
 - 11: Choose u.a.r. a size $\Delta_2 n'$ subset W_2 of V' and an assignment α_2 on W_2 ;
 (assume below that W_2 and α_2 are correctly chosen)
 - 12: $F'' \leftarrow F'[\alpha_2]$;
 - 13: Execute SparsePPSZ(F'')
-

Subprocedures² GetInd2Clauses, DensePPSZ $_p$, and SparsePPSZ that are used in HERTLI are stated as Algorithms 3, 4, and 5.

Based on [2,3]³ we can show that the following efficiency improvement is possible by HERTLI on uniquely satisfiable 3CNF formulas.

Theorem 1. *Use values given in the “value of [3]” column of Table 1 for the parameters of the procedure HERTLI and its subprocedures, and also for ε_0 . For any uniquely satisfiable 3CNF formula F , let E_{HERTLI} denote the event that HERTLI(F) yields the sat. assignment of F . Then we have $\log \Pr[E_{\text{HERTLI}}] \geq -(S - \varepsilon_0 + o(1))n$.*

We explain the proof of the theorem by showing that each procedure achieves its required task with desired probability. From now on till the end of this section,

² In [2], the part of the algorithm HERTLI corresponding to the statements 5–16 of Algorithm 2 is stated as algorithm OneCC (i.e., Algorithm 3 in [2]). On the other hand, we omit specifying it here and include it in Algorithm 2. In order to use algorithm numbering consistent with [2], we skip Algorithm 3 here and state GetInd2Clauses as Algorithm 4. While DensePPSZ $_p$ and SparsePPSZ correspond to procedures Dense (Algorithm 5) and Sparse (Algorithm 6) of [2], we modify their descriptions for the sake of our later explanation. As a whole, the procedure HERTLI is essentially the same as Hertli’s algorithm stated in [2].

³ Due to some minor error in [2], the choice of parameters in [2] is not appropriate, which has been corrected in [3]. Here we use this corrected version.

Algorithm 3. GetInd2Clauses **input:** a (≤ 3)CNF formula F'

- 1: $V' \leftarrow \text{vbl}(F')$; $n' \leftarrow |V'|$;
 - 2: $F_3 \leftarrow \{C \in F' : |C| = 3\}$; $F_2 \leftarrow \emptyset$; $W'_2 \leftarrow \emptyset$;
 - 3: **for** $T_2(n)$ times **do** (Define $T_2(n) = \lceil \Delta_2 n / 2 \rceil$.)
 - 4: $x \leftarrow$ a variable of F_3 with $\text{deg}_3(F_3, x) \geq 5$ (*failure stop* if no such variable exists)
 - 5: Choose C u.a.r. from all clauses of F_3 with $\ell(x) \in C$
 - 6: $C_2 \leftarrow C \setminus \{\ell(x)\}$
 - 7: $F_2 \leftarrow F_2 \cup \{C_2\}$; $W'_2 \leftarrow W'_2 \cup \text{vbl}(C_2)$
 - 8: $F_3 \leftarrow \{C \in F_3 : \text{vbl}(C) \cap \text{vbl}(C_2) = \emptyset\}$
 - 9: **end for**
 - 10: **return** F_2
-

Algorithm 4. DensePPSZ _{p}

input: a (≤ 3)CNF formula F' and a 2CNF formula F_2 , **parameter:** $p \in (0, 1)$

- 1: $V' \leftarrow \text{vbl}(F')$; $n' \leftarrow |V'|$;
 - 2: $V'_p \leftarrow$ pick each $x \in V'$ with probability p
 - 3: Let α'_2 be a partial assignment on V' initially undefined for all $x \in V'$
 - 4: **for** $C_2 \in F_2$ **do**
 - 5: **if** $\text{vbl}(C_2) \subseteq V_p$ **then** (let u and v are two literals of C_2)
 - 6: $(\alpha'_2(u), \alpha'_2(v)) \leftarrow \begin{cases} (0, 0) & \text{with probability } 1/5 (= 3/15), \text{ and} \\ (0, 1), (1, 0), \text{ or } (1, 1) & \text{with probability } 4/15 \text{ for each} \end{cases}$
 - 7: **end for**
 - 8: **for** $x \in V'_p$ **do**
 - 9: **if** $\alpha'_2(x)$ is undefined **then** $\alpha'_2(x) \leftarrow_{\text{u.a.r.}} \{0, 1\}$
 - 10: **end for**
 - 11: execute PPSZ($F'[\alpha'_2]$)
-

Algorithm 5. SparsePPSZ **input:** a (≤ 3)CNF formula F''

- 1: Let α'' be a partial assignment on F'' initially undefined for all $x \in \text{vbl}(F'')$
 - 2: $\tilde{F} \leftarrow F''$;
 - 3: **if** \tilde{F} has a unit clause **then** Extend α'' to satisfy all unit clauses and simplify \tilde{F}
(if an unsat. clause is derived in \tilde{F} during this step, then stop with “failure”)
 - 4: **while** \tilde{F} has some clause **do**
 - 5: $\tilde{V} \leftarrow \text{vbl}(\tilde{F})$; $\tilde{n} \leftarrow |\tilde{V}|$
 - 6: $F_2 \leftarrow \{C \in \tilde{F} : |C| = 2\}$
 - 7: **if** $|F_2| \leq \delta_3 \tilde{n}$ **then**
 - 8: Execute $\text{W_rand}(2\delta_3 + 4, \tilde{F})$
 - 9: **else**
 - 10: **assume** \tilde{F} has $\delta_4 \tilde{n}$ critical 2-clauses **then** Execute PPSZ(\tilde{F})
 - 11: **assume** otherwise **then**
 (that is, more than $1 - \delta_4 / \delta_3$ of 2-clauses of F_2 are noncritical)
 - 12: Choose C u.a.r. from F_2
 - 13: Extend α'' to satisfy all literals in C and simplify \tilde{F}
 - 14: **end while**
-

Table 1. Parameters used in Hertli’s algorithm and our improvements

Name	Reference	Name in [2]	Value in [3]	New value
ε_0	Theorem 1	ε_2	10^{-25}	$2.47 \cdot 10^{-19}$
ε_1	Lemma 7	ε_3	10^{-3}	$(2.32 \dots) \cdot 10^{-3}$
ε_2	Lemma 8	ε_1	10^{-20}	$(9.23 \dots) \cdot 10^{-15}$
Δ_1		Δ_1	10^{-22}	$1.6595 \cdot 10^{-16}$
Δ_2		Δ_2	$5 \cdot 10^{-5}$	$3.7736 \cdot 10^{-3}$
δ_3		–	1/10	0.159227
δ_4		–	1/30	0.06572
p	p^*		$5 \cdot 10^{-7}$	$(3.82 \dots) \cdot 10^{-5}$
q		–	–	$(10^5 \Delta_2 n)^{-1}$

we fix F (also V and n) to be any 3CNF formula with a unique sat. assignment α_* . Thus, by “success probability” we mean the probability that α_* is obtained. We assume that variables in the procedures with the same name are given these values and that parameters used in the procedures are set values given in the “value in [3]” column of Table 1. Values of this column are also used efficiency improvements ε_0 , ε_1 , and ε_2 . We also assume that n is quite large so that our choice of parameters would make sense.

First consider the outline of HERTLI. From Algorithm 2 we see that HERTLI uses three sat. algorithms for the following three cases:

- (H1) := [F has more than $\Delta_1 n$ variables with more than one critical clause]
- (H2) := [\neg (H1) $\wedge F'_*$ is Δ_2 -dense]
- (H3) := [\neg (H1) $\wedge F'_*$ is Δ_2 -sparse]

That is, PPSZ(F), DensePPSZ $_p(F'_*, F_{2,*})$, and SparsePPSZ(F''_*) are executed when (H1), (H2), and (H3) holds.

Here we consider the situation where the values of the variables W_1 , α_1 , W_2 , and α_2 of HERTLI are guessed “appropriately” and take the following values:

- $W_{1,*}$ = the set of variables with more than one critical clause,
- $W_{2,*}$ = a set of var.s of size $\leq 2T_2(n)$ s.t. $F \setminus W_{2,*}$ is 4-degree $_3$ bounded,
- $\alpha_{1,*}$ = $\alpha_*|_{W_{1,*}}$, and $\alpha_{2,*}$ = $\alpha_*|_{W_{2,*}}$

Then the variables F' , F'' , F_2 are set the following values: $F'_* = F[\alpha_{1,*}]$, $F''_* = F'_*[\alpha_{1,*}]$, and $F_{2,*} = \text{GetInd2Clauses}(F'_*)$, where the last one is for the case that $\text{GetInd2Clauses}(F'_*)$ successfully returns a result. We also use V'_* , n'_* , V''_* , and n''_* for the corresponding values, i.e., $\text{vbl}(F'_*)$, $|\text{vbl}(F'_*)|$, $\text{vbl}(F''_*)$, and $|\text{vbl}(F''_*)|$.

The case where (H1) holds is simple; in fact, we have already prepared Lemma 4 for this case, which gives the following success probability bound.

Lemma 6. *Suppose that (H1) holds for F . Then we have $\log \Pr[\mathbf{E}_{\text{PPSZ}}] \geq -(S - 0.00145 \cdots \Delta_1 + o(1))n$. That is, the log of the success probability of the line 2–3 of HERTLI is at least $-(S - 0.00145 \cdots \Delta_1 + o(1))n$, which is larger than $-(S - \varepsilon_0 + o(1))n$.*

Thus, for proving the theorem, it suffices to guarantee the efficiency improvement ε_0 for the other cases.

For the case where (H3) holds, the procedure SparsePPSZ is used. Its task is simply to get a sat. assignment for F'' given in this case. For its success probability, we have the following lemma, which is proved as Lemma 6 in [2]. Below we use $H(r)$ to denote the binary entropy, and use the well-known bound $\log \binom{n}{rn} \leq H(r)n$ that holds for any $r \in [0, 1]$ such that rn is an integer.

Lemma 7. *Suppose that (H3) holds for F . Let $\mathbf{E}_{\text{Sparse}}$ denote the event that SparsePPSZ(F'') returns $\alpha_*|V''$. Then we have $\log \Pr[\mathbf{E}_{\text{Sparse}}] \geq -(S - \varepsilon_1 + o(1))n''$. That is, the efficiency improvement of SparsePPSZ on F'' is at least ε_1 . Furthermore, including the probability of guessing $W_{1,*}$, $\alpha_{1,*}$, $W_{2,*}$, and $\alpha_{2,*}$, the log of the success probability of the line 10–13 of HERTLI is at least $-(S + \Delta_1 + H(\Delta_1) + \Delta_2 + H(\Delta_2) - \varepsilon_1)n$, which is larger than $-(S - \varepsilon_0 + o(1))n$.*

Finally consider the case where (H2) holds. In this case, HERTLI first executes GetInd2Clauses(F'_*) to get a set $F_{2,*}$ of $T_2(n)$ independent 2-clauses, and then executes DensePPSZ $_p(F'_*, F_{2,*})$ to get a sat. assignment for F'_* . Since (H2) holds, it is easy to see that the execution GetInd2Clauses(F'_*) always terminates successfully. Then 2-clauses of $F_{2,*}$ are obtained from 3-clauses of F'_* ; furthermore, it follows from (H2) that on average (w.r.t. the randomness of GetInd2Clauses) at least $4/5$ 2-clauses in $F_{2,*}$ are from noncritical 3-clauses of F'_* . This is a key to derive the following lower bound on the success probability of the execution DensePPSZ $_p(F'_*, F_{2,*})$.

Lemma 8. *Suppose that (H2) holds for F . Then GetInd2Clauses successfully returns a set of $T_2(n)$ independent 2-clauses. Let $\mathbf{E}_{\text{Dense}_p}$ denote the event that DensePPSZ $_p(F'_*, F_{2,*})$ returns $\alpha_*|V'_*$. Then we have $\log \Pr[\mathbf{E}_{\text{Dense}_p}] \geq -(S + I(p) - a_0 \Delta_2 p^2 + o(1))n'_*$, where $a_0 = 0.00505 \cdots$. That is, the efficiency improvement of DensePPSZ $_p$ on $(F'_*, F_{2,*})$ is at least $\varepsilon_2 := \max_p(-I(p) + a_0 \Delta_2 p^2)$. Thus, including the probability of guessing $W_{1,*}$ and $\alpha_{1,*}$, the log of the success probability of the line 7–9 of HERTLI is at least $-(S + \Delta_1 + H(\Delta_1) - \varepsilon_2 + o(1))n$, which is larger than $-(S - \varepsilon_0 + o(1))n$.*

3 Our Improvements

As explained in Introduction, the key idea of our main improvement is to use GetInd2Clauses for obtaining W_2 instead of guessing it randomly in the straightforward way, thereby removing the $-H(\Delta_2)$ term from the efficiency improvement of the sparse case (Lemma 7). In order to give a condition that this idea works, we introduce a “soft” version of the Δ -sparseness/-denseness.

Algorithm 6. newHERTLI**input:** a (≤ 3)CNF formula F , **parameter:** $\Delta_1, \Delta_2, \delta_3, \delta_4, p, q$

-
- 1: $V \leftarrow \text{vbl}(F)$; $n \leftarrow |V|$;
 - 2: **assume** F has more than $\Delta_1 n$ var.s with more than one critical clause **then**
 - 3: Execute PPSZ(F)
 - 4: **assume** otherwise **then**
 - 5: Choose u.a.r. a size $\Delta_1 n$ subset W_1 of V and an assignment α_1 on W_1 ;
 - 6: $F' \leftarrow F[\alpha_1]$; $V' \leftarrow \text{vbl}(F')$; $n' \leftarrow |V'|$
 - 7: **assume** F' is (Δ_2, q) -dense **then**
 - 8: $F_2 \leftarrow \text{GetInd2Clauses}(F')$
 - 9: Execute DensePPSZ $_p(F', F_2)$
 - 10: **assume** otherwise **then**
 - 11: $W_2 \leftarrow \text{GetInd2Clauses}_q^+(F')$
 - 12: Choose α_2 u.a.r. from all assignments on W_2 ;
 - 13: $F'' \leftarrow F'[\alpha_2]$;
 - 14: Execute SparsePPSZ(F'')
-

The new algorithm is given as Algorithm 6. It uses a procedure $\text{GetInd2Clauses}_q^+$ for computing a set W_2 corresponding to the one guessed in the original Hertli's algorithm. This procedure is obtained by modifying the procedure GetInd2Clauses on two points. For a given formula F' , instead of computing a set F_2 of independent 2-clauses, $\text{GetInd2Clauses}_q^+$ aims to compute a set W_2 such that $F' \setminus W_2$ becomes 4-degree $_3$ bounded. Thus, the line 4 of Algorithm 3 is modified so that if x cannot be found, then the algorithm stops *successfully* by reporting W_2 . On the other hand, the termination of its main for-loop, i.e., the line 3–9 part of Algorithm 3, is regarded as an undesired situation. $\text{GetInd2Clauses}_q^+$ tries this part for $1/q$ times for a given parameter q and stops with “failure” if a desired W_2 is not obtained by all trials.

Our new condition is to determine which of GetInd2Clauses and $\text{GetInd2Clauses}_q^+$ is likely to succeed. Consider the execution $\text{GetInd2Clauses}(F')$. We regard it as a random process of collecting an independent 2-clause from $F' \setminus W_2$ to F_2 while choosing a variables x with $\text{deg}_3(x) \geq 5$ randomly. For each $t \geq 1$, let N_t denote the event that there exists a degree $_3 \geq 5$ variable in F_3 at beginning of the t th iteration of the main for-loop and hence the algorithm executes the t th iteration. We define $N_{\leq t} \iff \bigwedge_{1 \leq i \leq t} N_i$ and $q_t = \Pr[\neg N_t \mid N_{\leq t-1}]$.

Definition 4. For any $q \in [0, 1]$ and $\Delta > 0$, a 3CNF formula F is (Δ, q) -dense if $q_t \leq q$ for all t , $1 \leq t \leq \lceil \Delta n/2 \rceil$; otherwise, F is (Δ, q) -sparse.

This is the new condition used in Algorithm 3. Although not exactly the same, the $(\Delta, 0)$ -dense/sparse condition is practically the same as the Δ -dense/sparse condition w.r.t. the execution of GetInd2Clauses .

Now our task is to show that the success probability of the new sparse case is in fact improved and that the success probability of the new dense case is not so affected. Similar to the previous discussion, we consider the execution of

the procedure newHERTLI when some sufficiently large and uniquely satisfiable 3CNF formula F is given as an input; the symbols such as F'_* , etc. are used in the same way as before while we leave the choice of parameter values for a later discussion. For a given parameter q , we define $(H2)_q^+$ and $(H3)_q^+$ by

$$\begin{aligned} (H2)_q^+ &:= [\neg(H1) \wedge F'_* \text{ is } (\Delta_2, q)\text{-dense}] \\ (H3)_q^+ &:= [\neg(H1) \wedge F'_* \text{ is } (\Delta_2, q)\text{-sparse}] \end{aligned}$$

We first show that the success probability is improved for the (Δ_2, q) -sparse case. In the following, let T_2 denote $T_2(n)$ and N denote the event $\bigwedge_{1 \leq i \leq T_2} N_i$.

Lemma 9. *Suppose that $(H3)_q^+$ holds for F . Then with $\Omega(1)$ probability W_2 is successfully computed by $\text{GetInd2Clauses}_q^+(F'_*)$. Thus, including the probability of guessing $W_{1,*}$, $\alpha_{1,*}$, and $\alpha_{2,*}$, the log of the success probability of the line 10–14 of newHERTLI is at least $-(S + \Delta_1 + H(\Delta_1) + \Delta_2 - \varepsilon_1)n$, where ε_1 is a lower bound for the efficiency improvement of SparsePPSZ on F''_* .*

Then by following lemma we can say that DensePPSZ works as well even under the condition $(H2)_q^+$.

Lemma 10. *Suppose that $(H2)_q^+$ holds for F . Then with probability at least $1 - T_2q$, $F_{2,*}$ is successfully computed by $\text{GetInd2Clauses}(F'_*)$. Recall that E_{Dense_p} denote the event that $\text{DensePPSZ}_p(F'_*, F_{2,*})$ returns $\alpha_*|_{V'_*}$. We have $\log \Pr[E_{\text{Dense}_p}] \geq -(S + I(p) - a_q \Delta_2 p^2 + o(1))n$, where*

$$a_q = \frac{1}{2} \left(2 + \log \left(\frac{4}{15} \right) - \frac{1}{5(1 - T_2q)} \log \left(\frac{4}{3} \right) \right),$$

which is close to a_0 of Lemma 8 by setting $q = (10^5 T_2)^{-1} = (10^5 \Delta_2 n)^{-1}$.

As stated Lemma 8, a lower bound for the efficiency improvement of $\text{DensePPSZ}_p(F'_*, F_{2,*})$ is calculated as $\varepsilon_2 := \max_p(-I(p) + a_q \Delta_2 p^2)$. Then including the probability of guessing $W_{1,*}$, $\alpha_{1,*}$, and $\alpha_{2,*}$, the log of the success probability of the line 7–9 of newHERTLI is at least $-\{\Delta_1 + H(\Delta_1) + (S - \varepsilon_2)(1 - \Delta_1)\}n$.

Now by setting the parameters used in the algorithms appropriately, we can show the following efficiency improvement.

Theorem 2. *For the procedure newHERTLI, use values given in the “new value” column of Table 1 for its parameters and also for ε_0 . For any uniquely satisfiable 3CNF formula, the log of its success probability is at least $-(S - \varepsilon_0 + o(1))n$.*

References

- Hertli, T.: 3-SAT faster and simpler-unique-SAT bounds for PPSZ hold in general. In: FOCS 2011, pp. 277–284 (2011)
- Hertli, T.: Breaking the PPSZ barrier for unique 3-SAT. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8572, pp. 600–611. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43948-7_50

3. Hertli, T.: Improved exponential algorithms for SAT and CISP. A thesis for Doctor of Sciences of ETH Zurich (2015)
4. Paturi, R., Pudlak, P., Saks, M.E., Zane, F.: An improved exponential-time algorithm for k -SAT. *J. ACM* **52**(3), 337–364 (2005)
5. Paturi, R., Pudlak, P., Zane, F.: Satisfiability coding lemma. *Chicago J. Theor. Comput. Sci.* 11–19 (1999)
6. Schönig, U.: A probabilistic algorithm for k -SAT and constraint satisfaction problems. In: *FOCS 1999*, pp. 410–414 (1999)
7. Scheder, D., Steinberger, J.P.: PPSZ for general k -SAT - making Hertli's analysis simpler and 3-SAT faster. In: *CCC 2017*, pp. 9:1–9:15 (2017)
8. Wahlström, M.: An algorithm for the SAT problem for formulae of linear length. In: Brodal, G.S., Leonardi, S. (eds.) *ESA 2005*. LNCS, vol. 3669, pp. 107–118. Springer, Heidelberg (2005). https://doi.org/10.1007/11561071_12

Random Popular Matchings with Incomplete Preference Lists

Suthee Ruangwises^(✉)  and Toshiya Itoh

Department of Mathematical and Computing Science,
Tokyo Institute of Technology, Yokohama, Japan
ruangwises.s.aa@m.titech.ac.jp, titoh@c.titech.ac.jp

Abstract. For a set A of n people and a set B of m items, with each person having a preference list that ranks some items in order of preference, we consider the problem of matching every person with a unique item. A matching M is *popular* if for any other matching M' , the number of people who prefer M to M' is not less than the number of those who prefer M' to M . For given n and m , consider the probability of existence of a popular matching when each person's preference list is independently and uniformly generated at random. Previously, Mahdian showed that when people's preference lists are *strict* (containing no ties) and *complete* (containing all items in B), if $\alpha = m/n > \alpha_*$, where $\alpha_* \approx 1.42$ is the root of equation $x^2 = e^{1/x}$, then a popular matching exists with probability $1 - o(1)$; and if $\alpha < \alpha_*$, then a popular matching exists with probability $o(1)$, i.e. a phase transition occurs at α_* . In this paper, we investigate phase transitions in more general cases when people's preference lists are not complete. In particular, we show that in the case that each person has a preference list of length k , if $\alpha > \alpha_k$, where $\alpha_k \geq 1$ is the root of equation $xe^{-1/2x} = 1 - (1 - e^{-1/x})^{k-1}$, then a popular matching exists with probability $1 - o(1)$; and if $\alpha < \alpha_k$, then a popular matching exists with probability $o(1)$.

Keywords: Popular matching · Incomplete preference lists
Phase transition · Complex component

1 Introduction

Consider the problem of matching people with items, with each person having a preference list that ranks some items in order of preference. This simple problem models many important real-world situations, such as the assignment of DVDs to subscribers [12], graduates to training positions [8], and families to government-subsidized housing [18].

The main target of such problems is to find the “optimal” matching in each situation. Various definitions of optimality have been proposed. The least restrictive one is *Pareto optimality* [1, 2, 16]. A matching M is Pareto optimal if there is no other matching M' such that at least one person prefers M' to M but

A full version of the paper is available at <https://arxiv.org/abs/1609.07288>.

no one prefers M to M' . Other stronger definitions include *rank-maximality* [9] (allocating maximum number of people to their first choices, then maximum number to their second choices, and so on), and *popularity* [3, 6] defined below.

1.1 Popular Matching

Consider a set A of n people and a set B of m items, with $\alpha = m/n$. Throughout this paper, we assume that $m \geq n$ and thus $\alpha \geq 1$. Each person has a preference list that ranks some items in order of preference. A preference list is *strict* if it does not contain ties, and is *complete* if it contains all items in B . We want to match every person with a unique item. In a matching M , for each person $a \in A$ and item $b \in B$, let $M(a)$ be an item matched with a , and $M(b)$ be a person matched with b (for convenience, let $M(a)$ be *null* for an unmatched person a).

Let $r_a(b)$ be the rank of item b in a 's preference list, with the most preferred item having rank 1, the second most preferred item having rank 2, and so on (for convenience, let $r_a(\text{null}) = \infty$). For any pair of matchings M and M' , we define $\phi(M, M')$ to be the number of people who prefer M to M' , i.e. $\phi(M, M') = |\{a \in A \mid r_a(M(a)) < r_a(M'(a))\}|$. We then define a matching M to *win* over a matching M' (and M' to *lose* to M) if there are more people who prefer M to M' than those who prefer M' to M , i.e. $\phi(M, M') > \phi(M', M)$. A *popular matching* is a matching that does not lose to any other matching. A popular matching may or may not exist, depending on the people's preference lists.

A probabilistic variant of this problem, the random popular matching problem, studies the probability that a popular matching exists in a random instance for each value of n and m , when each person's preference list is defined independently by selecting the first item $b_1 \in B$ uniformly at random, the second item $b_2 \in B \setminus \{b_1\}$ uniformly at random, the third item $b_3 \in B \setminus \{b_1, b_2\}$ uniformly at random, and so on.

1.2 Related Work

The concept of popularity of a matching was first introduced by Gardenfors [6] in the context of the stable marriage problem. Abraham et al. [3] presented the first polynomial time algorithm to find a popular matching in a given instance, or to report that none exists. The algorithm runs in $O(m+n)$ time when the preference lists contain no ties, and in $O(m\sqrt{n})$ time when the preference lists contain ties. Later, Mestre [15] generalized the algorithm to find a popular matching in the case that people are given different voting weights. That algorithm runs in $O(m+n)$ time when ties are not allowed, and in $O(\min(k\sqrt{n}, n)m)$ time when ties are allowed, where k is the number of distinct weights. A variant of this problem known as the capacitated house allocation problem allows an item to be matched with more than one person. Manlove and Sng [13] presented an algorithm to determine whether a popular matching exists in this setting. The algorithm runs in $O(\sqrt{C}n + L)$ time when ties are not allowed, and in $O((\sqrt{C} + n)L)$ time when ties are allowed, where C is the total capacity and L is the total length of people's preference lists. The notion of a popular matching also applies when the preference lists are two-sided (matching people with people), both

in the bipartite graph (marriage problem) and non-bipartite graph (roommates problem). Biró et al. [4] developed an algorithm to test popularity of a matching in these two settings and proved that determining whether a popular matching exists in these settings is an NP-hard problem when ties are allowed.

While a popular matching does not always exist, McCutchen [14] introduced two measures of the *unpopularity* of a matching, the unpopularity factor and the unpopularity margin, and showed that the problem of finding a matching that minimizes either measure is an NP-hard problem. Huang et al. [7] later gave algorithms to find a matching with bounded values of these measures in certain instances. Kavitha et al. [11] introduced the concept of a *mixed matching*, which is a probability distribution over matchings, and proved that a mixed matching that is popular always exists.

For the probabilistic variant of strict and complete preference lists, Mahdian [12] proved that if $\alpha = m/n > \alpha_*$, where $\alpha_* \approx 1.42$ is the root of equation $x^2 = e^{1/x}$, then a popular matching exists with high probability ($1 - o(1)$ probability) in a random instance. On the other hand, if $\alpha < \alpha_*$, a popular matching exists with low probability ($o(1)$ probability). The point $\alpha = \alpha_*$ can be regarded as a phase transition point, at which the probability rises from asymptotically zero to asymptotically one. Itoh and Watanabe [10] later studied the case when people are given two weights w_1, w_2 with $w_1 \geq 2w_2$, and found a phase transition at $\alpha = \Theta(n^{1/3})$.

1.3 Our Results

The probabilistic variant in the case that preference lists are not complete, with every person's preference list having the same length k , was mentioned and conjectured by Mahdian [12] and simulated by Abraham et al. [3], but the exact phase transition point, or whether it exists at all, had not been found yet. In this paper, we study that case and discover a phase transition at $\alpha = \alpha_k$, where $\alpha_k \geq 1$ is the root of equation $xe^{-1/2x} = 1 - (1 - e^{-1/x})^{k-1}$. In particular, we prove that for $k \geq 4$, if $\alpha > \alpha_k$, then a popular matching exists with high probability; and if $\alpha < \alpha_k$, then a popular matching exists with low probability. For $k \leq 3$, in which the equation does not have a solution in $[1, \infty)$, a popular matching always exists with high probability for every value of $\alpha \geq 1$.

2 Preliminaries

For convenience, we create a unique auxiliary *last resort item* ℓ_a for each person $a \in A$ and append ℓ_a to the end of a 's preference list, i.e. ℓ_a has lower preference than all other items in the list. By introducing the last resort items, we can assume that every person is matched because we can simply match any unmatched person a with ℓ_a . Note that these last resort items are not in B and do not count toward m , the total number of “real items”.

For each person $a \in A$, let $f(a)$ be the item at the top of a 's preference list. Let F be the set of items $b \in B$ such that there exists a person $a' \in A$ with $f(a') = b$, and let $S = B - F$. Then, for each person $a \in A$, let $s(a)$ be the highest ranked item in a 's preference list that is not in F . Note that $s(a)$ is well-defined for every $a \in A$ because of the existence of last resort items.

Definition 1. A matching M is A -perfect if every person $a \in A$ is matched with either $f(a)$ or $s(a)$.

Abraham et al. proved the following lemma, which holds for any instance with strict (not necessarily complete) preference lists.

Lemma 1 [3]. *In a given instance with strict preference lists, a popular matching exists if and only if an A -perfect matching exists.*

It is worth noting a simple but useful lemma about independent and uniform selection of items at random proved by Mahdian, which will be used throughout this paper.

Lemma 2 [12]. *Suppose that we pick y elements from the set $\{1, \dots, z\}$ independently and uniformly at random (with replacement). Let a random variable X be the number of elements in the set that are not picked. Then, $\mathbb{E}[X] = e^{-y/z}z - \Theta(1)$ and $\text{Var}[X] < \mathbb{E}[X]$.*

3 Complete Preference Lists Setting

We first consider the setting that every person's preference list is strict and complete. Note that when $m > n$ and the preference lists are complete, the last resort items are not necessary.

From a given instance, we construct a *top-choice graph*, a bipartite graph with parts B and S such that each person $a \in A$ corresponds to an edge connecting $f(a) \in B$ and $s(a) \in S$. Note that multiple edges are allowed in this graph. Previously, Mahdian proved the following lemma.

Lemma 3 [12]. *In a given instance with strict and complete preference lists, an A -perfect matching exists if and only if its top-choice graph does not contain a complex component, i.e. a connected component with more than one cycle.*

By Lemmas 1 and 3, the problem of determining whether a popular matching exists is equivalent to determining whether the top-choice graph contains a complex component. However, the difficulty is that the number of vertices in the randomly generated top-choice graph is not fixed. Therefore, a random bipartite graph $G(x, y, z)$ with fixed number of vertices is defined as follows to approximate the top-choice graph.

Definition 2. *For integers x, y, z , $G(x, y, z)$ is a bipartite graph with $V \cup U$ as a set of vertices, where $V = \{v_1, v_2, \dots, v_x\}$ and $U = \{u_1, u_2, \dots, u_y\}$. Each of the z edges of $G(x, y, z)$ is selected independently and uniformly at random (with replacement) from the set of all possible edges between a vertex in V and a vertex in U .*

This auxiliary graph has properties closely related to the top-choice graph. Mahdian then proved that if $\alpha > \alpha_* \approx 1.42$, then $G(m, h, n)$ contains a complex component with low probability for a range of values of h , and used those properties to conclude that the top-choice graph also contains a complex component with low probability, thus a popular matching exists with high probability.

Theorem 1 [12]. *In a random instance with strict and complete preference lists, if $\alpha > \alpha_*$, where $\alpha_* \approx 1.42$ is the solution of the equation $x^2 e^{-1/x} = 1$, then a popular matching exists with probability $1 - o(1)$.*

Theorem 1 serves as an upper bound of the phase transition point in the case of strict and complete preference lists. On the other hand, the following lower bound was also proposed by Mahdian along with a sketch of the proof, although the fully detailed proof was not given.

Theorem 2 [12]. *In a random instance with strict and complete preference lists, if $\alpha < \alpha_*$, then a popular matching exists with probability $o(1)$.*

4 Incomplete Preference Lists Setting

The previous section shows known results in the setting that preference lists are strict and complete. However, preference lists in many real-world situations are not complete, as people may regard only some items as acceptable for them.

In the setting that the preference lists are strict but not complete, we will consider the case that every person's preference list has equal length k (not counting the last resort item).

Definition 3. *For a positive integer $k \leq m$, an instance with k -incomplete preference lists is an instance with every person's preference list having length exactly k .*

Definition 4. *For a positive integer $k \leq m$, a random instance with strict and k -incomplete preference lists is an instance with each person's preference list is chosen independently and uniformly from the set of all $\frac{m!}{(m-k)!}$ possible k -permutations of the m items in B at random.*

Recall that $F = \{b \in B \mid \exists a' \in A, f(a') = b\}$ and for each person $a \in A$, $s(a)$ is the highest ranked item in a 's preference list not in F . The main difference from the complete preference lists setting is that, in the incomplete preference lists setting $s(a)$ can be either a real item or the last-resort item ℓ_a . For each person $a \in A$, let P_a be the set of items in a 's preference list (not including the last resort item ℓ_a). We then define $A_1 = \{a \in A \mid P_a \subseteq F\}$ and $A_2 = \{a \in A \mid P_a \not\subseteq F\}$. We have $s(a) = \ell_a$ if and only if $a \in A_1$.

4.1 Top-Choice Graph

Analogously to the complete preference lists setting, we define the top-choice graph of an instance with strict and k -incomplete preference lists to be a bipartite graph with parts B and $S \cup L$, where $L = \{\ell_a \mid a \in A\}$ is the set of last resort items. Each person $a \in A_2$ corresponds to an edge connecting $f(a) \in B$ and $s(a) \in S$. We call these edges *normal edges*. Each person $a \in A_1$ corresponds to an edge connecting $f(a) \in B$ and $s(a) = \ell_a \in L$. We call these edges *last resort edges*.

Although the statement of Lemma 3 proved by Mahdian [12] is for the complete preference lists setting, exactly the same proof applies to incomplete preference lists setting as well. The proof first shows that an A -perfect matching exists if and only if each edge in the top-choice graph can be oriented such that each vertex has at most one incoming edge (because if an A -perfect matching M exists, we can orient each edge corresponding to $a \in A$ toward the endpoint corresponding to $M(a)$, and vice versa). Then, the proof shows that for any top-choice graph H , each edge of H can be oriented in such manner if and only if H does not have a complex component. Thus we can conclude the following lemma.

Lemma 4. *In a given instance with strict and k -incomplete preference lists, an A -perfect matching exists if and only if its top-choice graph does not contain a complex component.*

In contrast to the complete preference lists setting, the top-choice graph in the incomplete preference lists setting has two types of edges with different distributions: normal edges and last resort edges, and cannot be approximated by $G(x, y, z)$ defined in the previous section. Therefore, we have to construct another auxiliary graph $G'(x, y, z_1, z_2)$ as follows.

Definition 5. *For integers x, y, z_1, z_2 , $G'(x, y, z_1, z_2)$ is a bipartite graph with $V \cup U \cup U'$ as a set of vertices, where $V = \{v_1, v_2, \dots, v_x\}$, $U = \{u_1, u_2, \dots, u_y\}$, and $U' = \{u'_1, u'_2, \dots, u'_{z_1+z_2}\}$. This graph has $z_1 + z_2$ edges. Each of the first z_1 edges is selected independently and uniformly at random (with replacement) from the set of all possible edges between a vertice in V and a vertice in U . Then, each of the next z_2 edges is constructed by the following procedures: Uniformly select a vertex v_i from V at random (with replacement); then, uniformly select a vertex u'_j that has not been selected before from U' at random (without replacement) and construct an edge (v_i, u'_j) .*

The intuition of $G'(x, y, z_1, z_2)$ is that we approximate the top-choice graph in the incomplete preference list setting, with V , U , and U' correspond to B , S , and L , respectively, and the first z_1 edges and the next z_2 edges correspond to normal edges and last resort edges, respectively.

Similarly to the complete preference lists setting, this auxiliary graph has properties closely related to the top-choice graph in incomplete preference lists setting, as shown in the following lemma. The proof of this lemma, which used the same technique as in Mahdian’s proof of [12, Lemma 3], is shown in the full version.

Lemma 5. *Suppose that $\alpha = m/n$, the top-choice graph H has t normal edges and $n - t$ last resort edges for a fixed integer $t \leq n$, and E is an arbitrary event defined on graphs. If the probability of E on the random graph $G'(m, h, t, n - t)$ is at most $O(1/n)$ for every fixed integer $h \in [e^{-1/\alpha}m - m^{2/3}, e^{-1/\alpha}m + m^{2/3}]$, then the probability of E on the top-choice graph H is at most $O(n^{-1/3})$.*

4.2 Size of A_2

Since our top-choice graph has two types of edges with different distributions, the first thing we want to bound is the number of each type of edges. Note that the top-choice graph has $|A_2|$ normal edges and $|A_1|$ last resort edges, so the problem is equivalent to bounding the size of A_2 .

We will prove the following lemma, which shows that in a random instance with strict and k -incomplete preference lists, the ratio $\frac{|A_2|}{n}$ lies around a constant $1 - (1 - e^{-1/\alpha})^{k-1}$ with high probability.

Lemma 6. *In a random instance with strict and k -incomplete preference lists,*

$$1 - (1 - e^{-1/\alpha})^{k-1} - c < \frac{|A_2|}{n} < 1 - (1 - e^{-1/\alpha})^{k-1} + c$$

with probability $1 - o(1)$ for any constant $c > 0$.

Proof. Let $c > 0$ be any constant. If $k = 1$, then we have $P_a \subseteq F$ for every $a \in A$, which means $|A_2| = 0$ and thus the lemma holds. From now on, we will consider the case that $k \geq 2$.

From Lemma 2, with $y = n$ and $z = m$, we have

$$\begin{aligned} \mathbb{E}[|F|] &= m - \mathbb{E}[|S|] = (1 - e^{-1/\alpha})m + \Theta(1); \\ \text{Var}(|F|) &= \text{Var}(|S|) < \mathbb{E}[|S|] < c_1 \mathbb{E}[|F|], \end{aligned}$$

for some constant $c_1 > 0$. Let $c' = \frac{c}{(k-1)(c+4)}$. By bounding the binomial expansions, we can verify that

$$(1 - e^{-1/\alpha} - c')^{k-1} > (1 - e^{-1/\alpha})^{k-1} - \frac{c}{4}; \quad (1)$$

$$(1 - e^{-1/\alpha} + c')^{k-1} < (1 - e^{-1/\alpha})^{k-1} + \frac{c}{4}. \quad (2)$$

Also, from Chebyshev's inequality we have

$$\Pr[||F| - \mathbb{E}[|F|]| \geq c' \cdot \mathbb{E}[|F|]] \leq \frac{\text{Var}[|F|]}{(c' \cdot \mathbb{E}[|F|])^2} \leq \frac{c_1}{c'^2 \cdot \mathbb{E}[|F|]} = O(1/n). \quad (3)$$

Let $I = [(1 - e^{-1/\alpha} - c')m, (1 - e^{-1/\alpha} + c')m]$. From (3) and the fact that $\mathbb{E}[|F|] = (1 - e^{-1/\alpha})m + \Theta(1)$, we have $|F| \in I$ with probability $1 - O(1/n) = 1 - o(1)$ for sufficiently large m .

Now suppose that $|F| = q$ for some fixed integer $q \in I$. For each $a \in A$, we have $a \in A_1$ if and only if $P_a - \{f(a)\} \subseteq F$. Consider that we first independently and uniformly select the first-choice item of every person in A from the set B at random, creating the set F . Then, for each $a \in A$, we uniformly select the remaining $k - 1$ items in a 's preference list one by one from the remaining $m - 1$ items

in $B - \{f(a)\}$ at random. Among the $(k - 1)\binom{m-1}{k-1}$ possible ways of selection, there are $(k - 1)\binom{q-1}{k-1}$ ways such that $P_a - \{f(a)\} \subseteq F$, so

$$\begin{aligned} \Pr [a \in A_1 | |F| = q] &= \Pr [P_a - \{f(a)\} \subseteq F | |F| = q] \\ &= \frac{(k - 1)\binom{q-1}{k-1}}{(k - 1)\binom{m-1}{k-1}} = \frac{\binom{q-1}{k-1}}{\binom{m-1}{k-1}}. \end{aligned}$$

Since $\binom{q-1}{k-1}/\binom{m-1}{k-1}$ converges to $(\frac{q}{m})^{k-1}$ when m becomes very large for every $q \in I$, it is sufficient to consider $\Pr [a \in A_1 | |F| = q] = (\frac{q}{m})^{k-1}$. Using this with (1) and (2), we can prove that

$$(1 - e^{-1/\alpha})^{k-1} - \frac{c}{2} < \Pr[a \in A_1] < (1 - e^{-1/\alpha})^{k-1} + \frac{c}{2},$$

where the detailed proof is given in the full version. This is equivalent to

$$1 - (1 - e^{-1/\alpha})^{k-1} - \frac{c}{2} < \Pr[a \in A_2] < 1 - (1 - e^{-1/\alpha})^{k-1} + \frac{c}{2}.$$

Finally, from this we can bound the expected value and variance of $|A_2|$, and use Chebyshev’s inequality to prove that

$$1 - (1 - e^{-1/\alpha})^{k-1} - c < \frac{|A_2|}{n} < 1 - (1 - e^{-1/\alpha})^{k-1} + c$$

with probability $1 - o(1)$, where the detailed proof is given in the full version. \square

5 Main Results

For each value of k , we want to find a phase transition point α_k such that if $\alpha > \alpha_k$, then a popular matching exists with high probability; and if $\alpha < \alpha_k$, then a popular matching exists with low probability. We do so by proving the upper bound and lower bound separately.

5.1 Upper Bound

Lemma 7. *Suppose that $\alpha = m/n$ and $0 \leq \beta < \alpha e^{-1/2\alpha}$. Then, the probability that $G'(m, h, \beta n, (1 - \beta)n)$ contains a complex component is at most $O(1/n)$ for every fixed integer $h \in [e^{-1/\alpha}m - m^{2/3}, e^{-1/\alpha}m + m^{2/3}]$.*

Proof. By the definition of $G'(m, h, \beta n, (1 - \beta)n)$, each vertex in U' has degree at most one, thus removing U' does not affect the existence of a complex component. Moreover, the graph $G'(m, h, \beta n, (1 - \beta)n)$ with part U' removed has exactly the same distribution as $G(m, h, \beta n)$ defined in Definition 2. Therefore, it is sufficient to consider the graph $G(m, h, \beta n)$ instead.

Using the same technique as in Mahdian’s proof of [12, Lemma 4], let X and Y be subsets of vertices of $G(m, h, \beta n)$ in V and U , respectively.

Define $BAD_{X,Y}$ to be an event that $X \cup Y$ contains either two vertices joined by three disjoint paths or two disjoint cycles joined by a path as a spanning subgraph. We call such subgraphs *bad* subgraphs. Note that every graph that contains a complex component must contain a bad subgraph. Then, let $p_1 = |X|$, $p_2 = |Y|$, and $p = p_1 + p_2$. Observe that $BAD_{X,Y}$ can occur only when $|p_1 - p_2| \leq 1$, so $p_1, p_2 \geq \frac{p-1}{2}$. Also, there are at most $2p^2$ non-isomorphic bad graphs with p_1 vertices in V and p_2 vertices in U , with each of them having $p_1!p_2!$ ways to arrange the vertices, and there are at most $(p+1)! \binom{\beta n}{p+1} \left(\frac{1}{mh}\right)^{p+1}$ probability that all $p+1$ edges of each graph are selected in our random procedure. So, the probability of $BAD_{X,Y}$ is at most

$$2p^2 p_1! p_2! (p+1)! \binom{\beta n}{p+1} \left(\frac{1}{mh}\right)^{p+1} \leq 2p^2 p_1! p_2! \left(\frac{\beta n}{mh}\right)^{p+1}.$$

By union bound, the probability that at least one $BAD_{X,Y}$ occurs is at most

$$\begin{aligned} \Pr \left[\bigvee_{X,Y} BAD_{X,Y} \right] &\leq \sum_{p_1, p_2} \binom{m}{p_1} \binom{h}{p_2} 2p^2 p_1! p_2! \left(\frac{\beta n}{mh}\right)^{p+1} \\ &\leq \sum_{p_1, p_2} \frac{m^{p_1}}{p_1!} \cdot \frac{h^{p_2}}{p_2!} \cdot 2p^2 p_1! p_2! \left(\frac{\beta}{\alpha h}\right)^{p+1} = \sum_{p_1, p_2} \frac{2p^2}{h} \left(\frac{\beta}{\alpha}\right)^{p+1} \left(\frac{m}{h}\right)^{p_1} \\ &\leq \sum_{p=1}^{\infty} \frac{O(p^2)}{n} \left(\frac{\beta}{\alpha}\right)^p \left(e^{-1/\alpha} - m^{-1/3}\right)^{-p/2} \\ &= \frac{O(1)}{n} \sum_{p=1}^{\infty} p^2 \left(\frac{\alpha^2}{\beta^2} \left(e^{-1/\alpha} - m^{-1/3}\right)\right)^{-p/2}. \end{aligned}$$

By the assumption, we have $\alpha^2 e^{-1/\alpha} > \beta^2$, so $\frac{\alpha^2}{\beta^2} (e^{-1/\alpha} - m^{-1/3}) > 1$ for sufficiently large m , thus the above sum converges. Therefore, the probability that at least one $BAD_{X,Y}$ happens is at most $O(1/n)$. \square

We can now prove the following theorem as an upper bound of α_k .

Theorem 3. *In a random instance with strict and k -incomplete preference lists, if $\alpha e^{-1/2\alpha} > 1 - (1 - e^{-1/\alpha})^{k-1}$, then a popular matching exists with probability $1 - o(1)$.*

Proof. Since $\alpha e^{-1/2\alpha} > 1 - (1 - e^{-1/\alpha})^{k-1}$, we can select a small enough $\delta_1 > 0$ such that $\alpha e^{-1/2\alpha} > 1 - (1 - e^{-1/\alpha})^{k-1} + \delta_1$. Let $J_1 = [(1 - (1 - e^{-1/\alpha})^{k-1} - \delta_1)n, (1 - (1 - e^{-1/\alpha})^{k-1} + \delta_1)n]$. From Lemma 6, $|A_2| \in J_1$ with probability $1 - o(1)$. Moreover, we have $\beta = \frac{t}{n} < \alpha e^{-1/2\alpha}$ for any integer $t \in J_1$.

Define E_1 to be an event that a popular matching exists in a random instance. First, consider the probability of E_1 conditioned on $|A_2| = t$ for each fixed integer $t \in J_1$. By Lemmas 5 and 7, the top-choice graph contains a complex component with probability $O(n^{-1/3}) = o(1)$. Therefore, from Lemmas 1 and 4

we can conclude that a popular matching exists with probability $1 - o(1)$, i.e. $\Pr[E_1 | |A_2| = t] = 1 - o(1)$ for every fixed integer $t \in J_1$. So

$$\begin{aligned} \Pr[E_1] &= \sum_t \Pr[|A_2| = t] \cdot \Pr[E_1 | |A_2| = t] \\ &\geq \sum_{t \in J_1} \Pr[|A_2| = t] \cdot \Pr[E_1 | |A_2| = t] \geq \Pr[|A_2| \in J_1] \cdot (1 - o(1)) \\ &= (1 - o(1))(1 - o(1)) = 1 - o(1). \end{aligned}$$

Thus a popular matching exists with probability $1 - o(1)$. □

5.2 Lower Bound

Lemma 8. *Suppose that $\alpha = m/n$ and $\alpha e^{-1/2\alpha} < \beta \leq 1$. Then, the probability that $G'(m, h, \beta n, (1 - \beta)n)$ does not contain a complex component is at most $O(1/n)$ for every fixed integer $h \in [e^{-1/\alpha}m - m^{2/3}, e^{-1/\alpha}m + m^{2/3}]$.*

Proof. Again, by the same reasoning as in the proof of Lemma 7, we can consider the graph $G(m, h, \beta n)$ instead of $G'(m, h, \beta n, (1 - \beta)n)$, but now we are interested in an event that $G(m, h, \beta n)$ does not contain a complex component.

Since $\alpha e^{-1/2\alpha} < \beta$, for sufficiently small $\epsilon > 0$, we still have $\alpha e^{-1/2\alpha} < (1 - \epsilon)^{3/2}\beta$. Consider the random bipartite graph $G(m, h, (1 - \epsilon)\beta n)$ with parts V having m vertices and U having h vertices. For each vertex v , let a random variable r_v be the degree of v . Since there are $(1 - \epsilon)\beta n$ edges in the graph, the expected value of r_v for each $v \in V$ is $c_1 = \frac{(1 - \epsilon)\beta n}{m} = \frac{(1 - \epsilon)\beta}{\alpha}$. Since $e^{-1/\alpha}m + m^{2/3} < \frac{\epsilon^{-1/\alpha}m}{1 - \epsilon}$ for sufficiently large m , the expected value of r_v for each $v \in U$ is

$$c_2 = \frac{(1 - \epsilon)\beta n}{h} > \frac{(1 - \epsilon)\beta n}{e^{-1/\alpha}m + m^{2/3}} > \frac{(1 - \epsilon)\beta n}{e^{-1/\alpha}m/(1 - \epsilon)} = \frac{(1 - \epsilon)^2\beta}{\alpha e^{-1/\alpha}}$$

for sufficiently large m . Furthermore, each r_v has a binomial distribution, which converges to Poisson distribution when m becomes very large. The graph can be viewed as a special case of *inhomogeneous random graph* [5, 17]. With the assumption that $c_1 c_2 > \frac{(1 - \epsilon)^3 \beta^2}{\alpha^2 e^{-1/\alpha}} > 1$, we can conclude that the graph contains a *giant component* (a component containing a constant fraction of vertices of the entire graph) with probability $1 - O(1/n)$, where the explanation is shown in the full version.

Finally, consider the construction of $G(m, h, \beta n)$ by putting $\epsilon\beta n$ more random edges into $G(m, h, (1 - \epsilon)\beta n)$. If two of those edges land in the giant component C , a complex component will be created. Since C has size of a constant fraction of m , each edge has a constant probability to land in C , so the probability that at most one edge will land in C is exponentially low. Therefore, $G(m, h, \beta n)$ does not contain a complex component with probability at most $O(1/n)$. □

We can now prove the following theorem as a lower bound of α_k .

Theorem 4. *In a random instance with strict and k -incomplete preference lists, if $\alpha e^{-1/2\alpha} < 1 - (1 - e^{-1/\alpha})^{k-1}$, then a popular matching exists with probability $o(1)$.*

Proof. Like in the proof of Theorem 3, we can select a small enough $\delta_2 > 0$ such that $\alpha e^{-1/2\alpha} < 1 - (1 - e^{-1/\alpha})^{k-1} - \delta_2$. Let $J_2 = [(1 - (1 - e^{-1/\alpha})^{k-1} - \delta_2)n, (1 - (1 - e^{-1/\alpha})^{k-1} + \delta_2)n]$. We have $\frac{|A_2|}{n} \in J_2$ with probability $1 - o(1)$ and $\beta = \frac{t}{n} > \alpha e^{-1/2\alpha}$ for any integer $t \in J_2$.

Now we define E_2 to be an event that a popular matching does not exist in a random instance. By the same reasoning as in the proof of Theorem 3, we can prove that $\Pr[E_2 | |A_2| = t] = 1 - o(1)$ for every fixed $t \in J_2$ and reach an analogous conclusion that $\Pr[E_2] = 1 - o(1)$. \square

5.3 Phase Transition

Since $f(x) = xe^{-1/2x} - (1 - (1 - e^{-1/x})^{k-1})$ is an increasing function in $[1, \infty)$ for every $k \geq 1$, $f(x) = 0$ can have at most one root in $[1, \infty)$. That root, if exists, will serve as a phase transition point α_k . In fact, for $k \geq 4$, $f(x) = 0$ has a unique solution in $[1, \infty)$; for $k \leq 3$, $f(x) = 0$ has no solution in $[1, \infty)$ and $\alpha e^{-1/2\alpha} > 1 - (1 - e^{-1/\alpha})^{k-1}$ for every $\alpha \geq 1$, so a popular matching always exists with high probability regardless of value of α without a phase transition. Therefore, from Theorems 3 and 4 we can conclude our main theorem below.

Theorem 5. *In a random instance with strict and k -incomplete preference lists with $k \geq 4$, if $\alpha > \alpha_k$, where $\alpha_k \geq 1$ is the root of equation $xe^{-1/2x} = 1 - (1 - e^{-1/x})^{k-1}$, then a popular matching exists with probability $1 - o(1)$; and if $\alpha < \alpha_k$, then a popular matching exists with probability $o(1)$. For $k \leq 3$, a popular matching always exists with probability $1 - o(1)$ in a random instance with k -incomplete preference lists for every $\alpha \geq 1$.*

5.4 Discussion

For each value of $k \geq 4$, the phase transition point occurs at the root $\alpha_k \geq 1$ of equation $xe^{-1/2x} = 1 - (1 - e^{-1/x})^{k-1}$ as shown in Fig. 1. Note

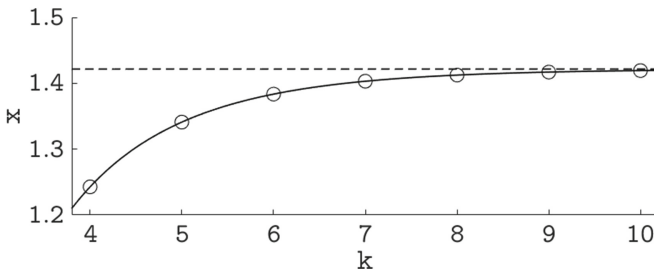


Fig. 1. Solution in $[1, \infty)$ of the equation $xe^{-1/2x} = 1 - (1 - e^{-1/x})^{k-1}$ for each $k \geq 4$, with the dashed line plotting $x = \alpha_* \approx 1.42$

that as k increases, the right-hand side of the equation converges to 1, thus α_k converges to Mahdian's value of $\alpha_* \approx 1.42$ in the case with complete preference lists.

References

1. Abdulkadiroğlu, A., Sönmez, T.: Random serial dictatorship and the core from random endowments in house allocation problems. *Econometrica* **66**(3), 689–701 (1998)
2. Abraham, D.J., Cechlárová, K., Manlove, D.F., Mehlhorn, K.: Pareto optimality in house allocation problems. In: Fleischer, R., Trippen, G. (eds.) *ISAAC 2004*. LNCS, vol. 3341, pp. 3–15. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30551-4_3
3. Abraham, D.J., Irving, R.W., Kavitha, T., Mehlhorn, K.: Popular matchings. In: *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 424–432 (2005)
4. Biró, P., Irving, R.W., Manlove, D.F.: Popular matchings in the marriage and roommates problems. In: Calamoneri, T., Diaz, J. (eds.) *CIAC 2010*. LNCS, vol. 6078, pp. 97–108. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13073-1_10
5. Bollobás, B., Janson, S., Riordan, O.: The phase transition in inhomogeneous random graphs. *Random Struct. Algorithms* **31**(1), 3–122 (2007)
6. Gärdenfors, P.: Match making: assignments based on bilateral preferences. *Behav. Sci.* **20**, 166–173 (1975)
7. Huang, C.-C., Kavitha, T., Michail, D., Nasre, M.: Bounded unpopularity matchings. In: Gudmundsson, J. (ed.) *SWAT 2008*. LNCS, vol. 5124, pp. 127–137. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69903-3_13
8. Hylland, A., Zeckhauser, R.: The efficient allocation of individuals to positions. *J. Polit. Econ.* **87**(22), 293–314 (1979)
9. Irving, R.W., Kavitha, T., Mehlhorn, K., Michail, D., Paluch, K.: Rank-maximal matchings. *ACM Trans. Algorithms* **2**(4), 602–610 (2006)
10. Itoh, T., Watanabe, O.: Weighted random popular matchings. *Random Struct. Algorithms* **37**(4), 477–494 (2010)
11. Kavitha, T., Mestre, J., Nasre, M.: Popular mixed matchings. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikolettseas, S., Thomas, W. (eds.) *ICALP 2009*. LNCS, vol. 5555, pp. 574–584. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02927-1_48
12. Mahdian, M.: Random popular matchings. In: *Proceedings of the 7th ACM Conference on Electronic Commerce (EC)*, pp. 238–242 (2006)
13. Manlove, D., Sng, C.T.S.: Popular matchings in the weighted capacitated house allocation problem. *J. Discrete Algorithms* **8**(2), 102–116 (2010)
14. McCutchen, R.M.: The least-unpopularity-factor and least-unpopularity-margin criteria for matching problems with one-sided preferences. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) *LATIN 2008*. LNCS, vol. 4957, pp. 593–604. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78773-0_51
15. Mestre, J.: Weighted popular matchings. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) *ICALP 2006*. LNCS, vol. 4051, pp. 715–726. Springer, Heidelberg (2006). https://doi.org/10.1007/11786986_62

16. Roth, A.E., Postlewaite, A.: Weak versus strong domination in a market with indivisible goods. *J. Math. Econ.* **4**, 131–137 (1977)
17. Söderberg, B.: General formalism for inhomogeneous random graphs. *Phys. Rev. E* **66**(6), 066121 (2002)
18. Yuan, Y.: Residence exchange wanted: a stable residence exchange problem. *Eur. J. Oper. Res.* **90**, 536–546 (1996)

Scheduling Batch Processing in Flexible Flowshop with Job Dependent Buffer Requirements: Lagrangian Relaxation Approach

Hanyu Gu, Julia Memar^(✉), and Yakov Zinder

University of Technology Sydney, PO Box 123, Broadway, NSW 2007, Australia
{hanyu.gu,julia.memar,yakov.zinder}@uts.edu.au

Abstract. The paper presents a Lagrangian relaxation based algorithm for scheduling jobs in the two-stage flowshop where the first stage is comprised of several parallel identical machines and the second stage consists of a single machine processing jobs in the predefined batches. Motivated by applications where unloading and loading occur when the means of transportation are changed, the processing of the jobs, constituting a batch, can commence only if this batch has been allocated a portion of a limited buffer associated with the flowshop. This portion varies from batch to batch and is released only after the completion of the batch processing on the second stage machine. Each batch has a due date and the objective is to minimise the total weighted tardiness. The effectiveness of the proposed algorithm is demonstrated by computational experiments.

Keywords: Scheduling · Flexible flowshop · Weighted total tardiness
Limited buffer · Lagrangian relaxation

1 Introduction

This paper is motivated by various systems where the change of the means of transportation requires the considerable storage space (referred below as a buffer). For example, in the case of supply chains of mineral resources, the material is transported by a fleet of trains or trucks and is stored as stockpiles on the so called pads prior to loading for the second stage of transportation. At this second stage the stockpiles are transported in groups. It is common to reserve the entire space for all stockpiles of a group prior to the arrival of the first load for these stockpiles and to release this space only after the completion of loading all stockpiles of the group for further transportation [5].

Another source of motivation is various computerised systems where files are to be loaded prior to their processing in the batch mode. Here, the computer memory can be viewed as a buffer. Analogously to the mentioned above transportation systems, in such computerised systems the portion of the buffer, required for all files that are to be processed as one batch, should be reserved prior to the loading the first file of the batch and is released only after the completion of this batch [10, 11].

This can be modelled as the two-stage flexible flowshop with a buffer of limited capacity, where each job is processed first on one of the parallel identical machines (first stage) and then by the single machine (second stage). The set of all jobs is partitioned into batches. This partition is predefined and it is a part of input. The partition is applicable to the second stage only, i.e. the second stage machine processes jobs in these predefined batches. The second stage machine can process at most one batch at a time.

The first stage operation of a job can be assigned to any of the first stage parallel identical machines. Each of these machines can process at most one job at a time. The first stage operations of all jobs, constituting a batch, must be completed before the start of the batch processing on the second stage machine.

Each batch seizes a portion of the buffer from the start of processing the jobs, constituting this batch, on the first stage machines till the completion of the processing this batch on the second stage machine. The portion of the buffer, seized by the batch, remains the same during this entire period and varies from batch to batch.

Although it is well known that the scheduling models, where the buffer requirements vary from job to job, better reflect numerous practical situations in comparison with the models where the buffer just limits the number of jobs [15], the literature on flowshop scheduling focuses almost entirely on the latter models [1, 2, 14]. Furthermore, most of these publications consider only buffers that restrict number of jobs that completed one stage of processing and are waiting for the next stage and ignore the important situation when a job occupies the buffer for the whole period of its processing, including the time between completion of one operation and the start of the next one. To the authors knowledge, only few publications address this gap in the literature on flowshop scheduling [4, 7–13]. This paper contributes to this efforts as follows:

- by studying a two-stage flexible flowshop with batch processing on the second stage that, to the authors' knowledge, has been never considered previously;
- by considering the problem of minimising the total weighted tardiness - the objective that, to the authors' knowledge, never has been studied for the models with job dependent buffer requirements;
- by presenting a new Lagrangian based algorithm together with the results of computational experiments that demonstrate its effectiveness.

The considered scheduling problem is strongly *NP*-hard because even its particular case with no buffer, no batches, and the objective of the total completion time is strongly *NP*-hard [6].

It also can be viewed as a resource constrained scheduling problem [1], but in contrast to the common resource constrained scheduling models, where the additional resource is used only when a machine processes an operation, in this paper the resource (buffer) is used from the start of processing the jobs, constituting a batch, on the first stage machines till the completion of the processing this batch on the second stage machine.

The rest of the paper is organised as follows. Section 2 introduces notation and an integer programming formulation. The Lagrangian relaxation approach is described in Sect. 3. The proposed Lagrangian heuristic is discussed in Sect. 4. The results of computational experiments are presented in Sect. 5. Conclusions are outlined in Sect. 6.

2 Notation and Integer Programming Formulation

Let $N = \{1, \dots, n\}$ be the set of jobs that are to be processed; n_b be the number of batches; N_k be the set of jobs in batch k , $1 \leq k \leq n_b$; and M be the number of parallel identical first stage machines. Denote by p_i the processing time of job i on a first stage machine; by ρ_k the processing time of batch k . All processing times are integer. Each batch k has the associated due date d_k and weight w_k , and requires $b(k)$ units of buffer capacity which it seises from the start of processing of the first job of the batch till the batch completion by the second stage machine. At any time, the total buffer requirement cannot exceed the buffer capacity B .

Assume that processing of jobs commences at time $t = 0$. A schedule is two sets, $\{S_1^1, \dots, S_n^1\}$ and $\{S_1^2, \dots, S_{n_b}^2\}$, where S_i^1 is the time when a first stage machine starts processing job i and S_k^2 is the time when the second stage machine starts processing batch k . The goal is to find a schedule with the smallest value of total weighted tardiness $\sum_{k=1}^{n_b} w_k T_k$, where $C_k = S_k^2 + \rho_k$ is the completion time of batch k and $T_k = \max\{0, C_k - d_k\}$ is the tardiness of batch k .

2.1 Integer Programming Formulation

Let integer T be the planning horizon, i.e. $C_k \leq T$ for all batches. For each job $i \in N$ and integer index $t \in [0, T)$, define

$$x_{it} = \begin{cases} 1, & \text{if } S_i^1 = t \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

For each batch $1 \leq k \leq n_b$ and integer index $t \in [0, T)$, define

$$y_{kt} = \begin{cases} 1, & \text{if } S_k^2 = t \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

For each batch $1 \leq k \leq n_b$ and integer index $t \in [0, T)$, define

$$z_{kt} = \begin{cases} 1, & \text{if } \sum_{i \in N_k} \sum_{\tau=0}^t x_{i\tau} > 0 \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

In other words, $z_{kt} = 1$ if there exists $i \in N_k$ such that $S_i^1 \leq t$.

Then, the considered scheduling problem can be formulated as the following integer linear program:

$$\min \sum_{k=1}^{n_b} w_k T_k, \quad (4)$$

subject to

$$\sum_{t=0}^{T-1} x_{it} = 1, \quad \text{for } 1 \leq i \leq n \quad (5)$$

$$\sum_{i=1}^n \sum_{\tau=\max\{0, t-p_i+1\}}^t x_{i\tau} \leq M, \quad \text{for } 0 \leq t < T \quad (6)$$

$$\sum_{i \in N_k} \sum_{\tau=0}^t x_{i\tau} - z_{kt} |N_k| \leq 0, \quad \text{for } 0 \leq t < T, \quad 0 \leq k \leq n_b \quad (7)$$

$$\sum_{i \in N_k} \sum_{\tau=0}^t x_{i\tau} - z_{kt} \geq 0, \quad \text{for } 0 \leq t < T, \quad 0 \leq k \leq n_b \quad (8)$$

$$\sum_{t=0}^{T-1} t(y_{kt} - x_{it}) \geq p_i, \quad \text{for } i \in N_k, \quad 0 \leq k \leq n_b \quad (9)$$

$$\sum_{k=1}^{n_b} b(k) \left(z_{kt} - \sum_{\tau=0}^{t-\rho_k} y_{k\tau} \right) \leq B, \quad \text{for } 0 \leq t < T \quad (10)$$

$$\sum_{k=1}^{n_b} \sum_{\tau=\max\{0, t-\rho_k+1\}}^t y_{k\tau} \leq 1, \quad \text{for } 0 \leq t < T \quad (11)$$

$$\sum_{t=0}^{T-1} y_{kt} = 1, \quad \text{for } 1 \leq k \leq n_b \quad (12)$$

$$T_k \geq \sum_{t=0}^{T-1} t y_{kt} + \rho_k - d_k, \quad \text{for } 1 \leq k \leq n_b \quad (13)$$

$$T_k \geq 0, \quad x_{it}, y_{kt}, z_{kt} \in \{0, 1\}, \quad \text{for } i \in N, 0 \leq t < T, 1 \leq k \leq n_b \quad (14)$$

The constraints (5) and (12) guarantee, that a job or a batch starts only once on the first or on the second stage, correspondingly; (6) and (11) are capacity constraints for the first and the second stage, correspondingly; (7) and (8) define value of z_{kt} , which is used for the buffer capacity constraint (10). The constraint (9) prevent a batch to start processing on the second stage before all jobs of the batch have been processed on the first stage. The constraint (13) together with $T_k \geq 0$ introduce tardiness T_k .

3 Lagrangian Relaxation

The Lagrangian Relaxation is obtained by dualizing (6), (10) and (11) for the chosen nonnegative Lagrange multipliers v_t , u_t and q_t , where $0 \leq t < T$:

$$\begin{aligned}
& \min \sum_{k=1}^{n_b} w_k T_k + \sum_{\tau=0}^{T-1} v_\tau \left(\sum_{i=1}^n \sum_{\lambda=\max\{0, \tau-p_i+1\}}^{\tau} x_{i\lambda} - M \right) \\
& + \sum_{\tau=0}^{T-1} u_\tau \left(\sum_{k=1}^{n_b} b(k) \left(z_{k\tau} - \sum_{\lambda=0}^{\tau-\rho_k} y_{k\lambda} \right) - B \right) \\
& + \sum_{\tau=0}^{T-1} q_\tau \left(\sum_{k=1}^{n_b} \sum_{\lambda=\max\{0, \tau-\rho_k+1\}}^{\tau} y_{k\lambda} - 1 \right) \tag{15}
\end{aligned}$$

subject to (5), (7)–(9) and (12)–(14). Denote by v , u and q the sets of all v_t , u_t and q_t , correspondingly. The problem above can be decomposed into n_b separate subproblems as follows. For each $1 \leq k \leq n_b$ let $Z_k(v, u, q)$ be the optimal value of the objective function of the integer linear program:

$$\begin{aligned}
& \min w_k T_k + \sum_{i \in N_k} \sum_{\tau=0}^{T-1} v_\tau \sum_{\lambda=\max\{0, \tau-p_i+1\}}^{\tau} x_{i\lambda} \\
& + b(k) \sum_{\tau=0}^{T-1} u_\tau \left(z_{k\tau} - \sum_{\lambda=0}^{\tau-\rho_k} y_{k\lambda} \right) + \sum_{\tau=0}^{T-1} q_\tau \sum_{\lambda=\max\{0, \tau-\rho_k+1\}}^{\tau} y_{k\lambda} \tag{16}
\end{aligned}$$

subject to

$$\sum_{t=0}^{T-1} x_{it} = 1, \quad \text{for } i \in N_k \tag{17}$$

$$\sum_{i \in N_k} \sum_{\tau=0}^t x_{i\tau} - z_{kt} |N_k| \leq 0, \quad \text{for } 0 \leq t < T \tag{18}$$

$$\sum_{i \in N_k} \sum_{\tau=0}^t x_{i\tau} - z_{kt} \geq 0, \quad \text{for } 0 \leq t < T \tag{19}$$

$$\sum_{t=0}^{T-1} t(y_{kt} - x_{it}) \geq p_i, \quad \text{for } i \in N_k \tag{20}$$

$$\sum_{t=0}^{T-1} y_{kt} = 1 \tag{21}$$

$$T_k \geq \sum_{t=0}^{T-1} t y_{kt} + \rho_k - d_k \tag{22}$$

$$T_k \geq 0, x_{it} \in \{0, 1\}, y_{kt} \in \{0, 1\}, z_{kt} \in \{0, 1\}, \quad \text{for } i \in N_k, 0 \leq t < T \tag{23}$$

Let $LR(v, u, q)$ be the optimal value of the objective function (15). As values $Z_k(v, u, q)$ are independent from each other, $LR(v, u, q)$ is a sum of $Z_k(v, u, q)$ and a linear combination of parameters:

$$LR(v, u, q) = \sum_{k=1}^{n_b} Z_k(v, u, q) - \sum_{t=0}^{T-1} (Mv_t + Bu_t + q_t) \quad (24)$$

Therefore, the Lagrangian Relaxation can be solved for the chosen Lagrangian multipliers by solving n_b separate integer linear programs (16)–(23).

In order to update the sets of Lagrangian multipliers v , u and q to maximise $LR(v, u, q)$, i.e. to solve the Lagrangian dual problem, we use the standard sub-gradient method [3], where at each iteration the solution of the current Lagrange Relaxation problem and the smallest current value of the objective function provided by a feasible schedule are used to obtain the new set of the multipliers for the next iteration.

3.1 Solution of the Lagrangian Relaxation Subproblems

Each of the n_b Lagrangian Relaxation subproblems can be solved with the efficient technique described below. Assume that a batch k starts processing at time s and completes at time $t + \rho_k$. This implies, that the batch completes on the first stage by time t . Then the objective function (16) can be presented as follows:

$$\begin{aligned} \min w_k T_k + \sum_{i \in N_k} \sum_{\tau=s}^{t-1} v_\tau \sum_{\lambda=\max\{s, \tau - p_i + 1\}}^{\tau} x_{i\lambda} \\ + b(k) \sum_{\tau=s}^{t+\rho_k-1} u_\tau + \sum_{\tau=t}^{t+\rho_k-1} q_\tau \sum_{\lambda=\max\{t, \tau - \rho_k + 1\}}^{\tau} y_{k\lambda} \end{aligned} \quad (25)$$

Denote by $g(s, t)$ the following

$$g(s, t) = w_k T_k + b(k) \sum_{\tau=s}^{t+\rho_k-1} u_\tau + \sum_{\tau=t}^{t+\rho_k-1} q_\tau,$$

and denote by $f(s, t)$

$$f(s, t) = \min \sum_{i \in N_k} \sum_{\tau=s}^{t-1} v_\tau \sum_{\lambda=\max\{s, \tau - p_i + 1\}}^{\tau} x_{i\lambda} \quad (26)$$

Taking into account that $\sum_{\lambda=\max\{0, \tau - \rho_k + 1\}}^{\tau} y_{k\lambda} = 1$ for all $t \leq \tau \leq t + \rho_k - 1$, (25) can be presented as

$$f(s, t) + g(s, t) \quad (27)$$

For job $i \in N_k$ processed on the first stage within the interval $[s, t]$ and assuming that $t - s \geq p_i$, denote by $f_i(s, t)$ the following:

$$f_i(s, t) = \min_{x_{i\tau}=1; \tau=s, \dots, t-p_i} \sum_{\lambda=s}^{t-1} v_\lambda \sum_{\alpha=\max\{0, \lambda-p_i+1\}}^{\lambda} x_{i\alpha}$$

Observe that $\sum_{\alpha=\max\{0, \lambda-p_i+1\}}^{\lambda} x_{i\alpha} = 1$ only for $\tau \leq \lambda \leq \tau + p_i - 1$. Hence

$$f_i(s, t) = \min_{\tau=s, \dots, t-p_i} \sum_{\lambda=\tau}^{\tau+p_i-1} v_\lambda$$

Observe that $f_i(s, s + p_i) = \sum_{\lambda=s}^{s+p_i-1} v_\lambda$ and

$$f_i(s, t + 1) = \min\{f_i(s, t), \sum_{\lambda=t-p_i+1}^t v_\lambda\}$$

Then $f(s, t)$ can be presented as follows:

$$f(s, t) = \min_{i \in N_k} \left\{ \sum_{\lambda=s}^{s+p_i-1} v_\lambda + \sum_{j \neq i; j \in N_k} f_j(s, t) \right\}$$

Then

$$Z_k(v, u, q) = \min_{[s, t]} (f(s, t) + g(s, t)),$$

where $0 \leq s < s + \rho_k \leq t \leq T - \rho_k$.

3.2 Choice of the Planning Horizon

The planning horizon T has a significant impact on the efficiency of the solution method for the Lagrangian Relaxation problem. A naive upper bound for T can be estimated as $T \leq \sum_{i \in N} p_i + \sum_{1 \leq k \leq n_b} \rho_k$. However, this upper bound of the planning horizon can be tightened which is beneficial to the performance of the Lagrangian Relaxation solution approach.

Let σ^* be an optimal schedule for criterion of total weighted tardiness. We assume that σ^* is an active schedule.

Definition 1. *A time interval is incomplete, if during the entire interval at least one machine is idle on the first stage at any point of time, and there are no batches processed on the second stage.*

Definition 2. *A time interval is full, if during the entire interval there are no idle machines on the first stage at any point of time, and there are no batches processed on the second stage.*

Definition 3. A time interval is loaded, if during the entire interval there is a batch processed on the second stage at any point of time.

Lemma 1. For any incomplete time interval there is at least one machine on the first stage which is idle during the entire time interval.

Proof. Consider the machine on the first stage which is idle at the start of the interval. If any job starts on this machine during the interval, it would imply that either the schedule is not active and the job could start earlier, or that there was not enough space in the buffer before the start of the job, hence a batch had to be released from the buffer immediately before the job started. This would imply that the batch has been processed during the interval, which contradicts to the interval being incomplete. \square

Definition 4. Let the list of batches be constructed in increasing order of batches' completion times. An incomplete time interval is canonically partitioned, if it is partitioned into subintervals such that for each subinterval there is a job selected as follows. Among all jobs which are processed during the entire subinterval, select a job from the batch on the earliest position on the list of batches. The selected job is a canonical cover of the subinterval.

Theorem 1. The set of canonical covers of all incomplete intervals contains at most one job from each batch.

Proof. Assume that for the incomplete intervals $[t_1, t_2]$ and $[t_3, t_4]$, $t_2 \leq t_3$, the corresponding canonical covers are jobs i_1 and i_2 , and both jobs are from the same batch j and $S_{i_2}^1 > t_1$. If the intervals are consecutive, i.e. $t_2 = t_3$, then by virtue of Lemma 1, there is an idle machine on the first stage for the entire time interval $[t_1, t_2]$, hence i_2 could start earlier, which contradicts to the schedule being active.

Assume that the intervals are not consecutive, i.e. $t_2 < t_3$. Then there is at least one machine from either stage is not idle on the interval $[t_2, t_3]$, otherwise the schedule would not be active. Assume that a batch k starts on second stage at time S_k^2 , and $t_2 \leq S_k^2 < t_3$. There are two possibilities: either the second stage machine is idle before batch k starts and there is a job h from the batch k such that the completion time of the job $C_h = S_k^2$ and the starting time of job h $S_h^1 > t_2$; or there are batches scheduled on the second stage before k which prevent k to start earlier. Observe, that since both intervals are incomplete, the batches will have to start and complete during the interval $[t_2, t_3]$.

In the former case, if $S_h^1 \leq t_1$, then h had to be selected as the canonical cover for the interval $[t_1, t_2]$, as batch k completes before batch j . If $S_h^1 > t_1$, then by virtue of Lemma 1 there is an idle machine at time t_1 on the first stage, hence h could have started earlier, which contradicts to the schedule being active. In the latter case, select the batch which started before k with the earliest starting time such that the second stage machine is idle before the batch starts on second stage and repeat the reasoning for the former case.

Since both intervals are incomplete, there is no batch processed on the entire interval $[t_1, t_4]$. Hence there is at least one machine is not idle on the first stage

during the interval $[t_2, t_3]$. If there is a full interval within $[t_2, t_3]$, then some batch must have completed on second stage to release buffer space immediately before the full interval started, otherwise by virtue of Lemma 1 some job scheduled during the full interval could have started earlier on an idle machine. However we have shown above that there is no batch processed $[t_1, t_4]$. Hence the only option left to consider is that $[t_2, t_3]$ is an incomplete interval. Similar to the above, in this case by virtue of Lemma 1 there is an idle machine on the first stage on both intervals $[t_1, t_2]$ and $[t_2, t_3]$, and hence i_2 could have started earlier, which contradicts to the schedule being active. \square

Let I, F and L be the total length of incomplete, full an loaded time intervals, correspondingly, in schedule σ^* . Then taking into account the Theorem 1,

$$\begin{aligned}
& \max_{1 \leq k \leq n_b} C_k(\sigma^*) = L + F + I \\
& \leq \sum_{1 \leq k \leq n_b} \rho_k + \frac{\sum_{i \in N} p_i - I}{M} + I = \sum_{1 \leq k \leq n_b} \rho_k + \frac{\sum_{i \in N} p_i}{M} + (1 - \frac{1}{M})I \\
& \leq \sum_{1 \leq k \leq n_b} \rho_k + \frac{\sum_{i \in N} p_i}{M} + (1 - \frac{1}{M}) \sum_{1 \leq k \leq n_b} \max_{i \in N_k} p_i \tag{28}
\end{aligned}$$

Hence the upper bound for planning horizon T can be set to the value of (28).

4 Lagrangian Heuristic

The optimal solution obtained by Lagrangian Relaxation for given Lagrangian multipliers, provides the starting times of jobs on the first stage machines and batches on both - the first and the second stages. The schedule, specified by these starting times, is often not feasible, so the proposed Lagrangian Heuristic uses the orders defined by these starting times, to construct a feasible schedule and perhaps improve upper bound for the objective function. Denote by π_1 and π_2 the permutations specified by the starting times of the batches on the first and the second stage, correspondingly. For each batch k denote by π^k the permutation of jobs in the batch specified by job's starting times. In [7] we considered two-stage flowshop with independent buffers and proposed a few Lagrangian Heuristics, which are based on "no-wait" and "wait" approach. The "no-wait" Lagrangian Heuristic can violate the permutations, whereas the "wait" heuristic strictly follows these permutations. The "wait" heuristic which followed π_1 on both stages, provided the best values of objective function for most of test instances. The algorithm below is based on this heuristic.

In the proposed algorithm we assume that on both stages the schedule is defined by the same order π determined by the order of batches on the first stage, and the algorithm processes batches strictly in this order on each stage. It is easy to see that for any permutation π such a schedule exists. Similar to [7], consider batch j on position $\pi^{-1}(j)$ in π . If total buffer requirement of batches in positions before and including $\pi^{-1}(j)$ does not exceed the buffer capacity,

then all these batches can be processed on the first stage in order of π without violating the buffer capacity. If for batch j

$$\sum_{1 \leq u \leq \pi^{-1}(j)} b(\pi(u)) > B, \quad (29)$$

then we will determine the smallest position k_j in π such that

$$\sum_{1 \leq u \leq \pi^{-1}(j)} b(\pi(u)) - \sum_{1 \leq u \leq k_j} b(\pi(u)) \leq B. \quad (30)$$

It is easy to see that $k_j < \pi^{-1}(j)$, and hence in order to have enough buffer space for j to start on the first stage, it is sufficient to wait till all the batches i which were placed on the first stage before and including position k_j : $\pi^{-1}(i) \leq k_j$, are processed on the second stage. We refer to each batch j , for which (29) holds, as to *ordinary* batch.

4.1 Wait Algorithm

Denote by t_1 and t_2 the current minimal starting time available for unscheduled batches on the first and the second stage, correspondingly. Denote by pos_1 and pos_2 the current position in π on first and the second stages, correspondingly. Let τ_m be the current minimal starting time available for unscheduled jobs on machine m of the first stage, $1 \leq m \leq M$. Set all τ_m , t_1 and t_2 to zero and set $pos_1 = pos_2 = 1$.

First stage:

- if all batches are scheduled on the first stage, go to *Second stage* step;
- if batch j on the current position pos_1 is not ordinary batch or the batch $q = \pi^{-1}(k_j)$ is scheduled and $S_q^2 + \rho_2 \leq t_1$, then set the starting times to jobs $i \in N_j$, assigning job i in the order specified by π^j to the machine m with smallest τ_m : $S_i^1 = \tau_m$; change $\tau_m = \tau_m + p_i$. Once all jobs from batch j have been scheduled, set $t_1 = \min_{1 \leq m \leq M} \tau_m$ and the completion time of batch j on the first stage $C_j^1 = \max_{i \in N_j} (S_i^1 + p_i)$; increase the current position pos_1 by one, and go to *Second stage* step;
- if for the ordinary batch j on the current position pos_1 the batch $q = \pi^{-1}(k_j)$ is scheduled but $S_q^2 + \rho_2 > t_1$, then set $t_1 = S_q^2 + \rho_2$, and for all $1 \leq m \leq M$, set $\tau_m = \max\{\tau_m, t_1\}$; go to the previous step to schedule batch j on the first stage;
- if for the ordinary batch j on the current position pos_1 the batch $q = \pi^{-1}(k_j)$ has not been scheduled, go to *Second stage* step.

Second stage:

- if all batches are scheduled on the second stage, stop;
- if for the batch j on the current position pos_2 its completion time on the first stage $C_j^1 \leq t_2$, then assign $S_j^2 = t_2$, set $t_2 = S_j^2 + \rho_j$, increase the current position pos_2 by one, and go to *First stage* step;
- if for the batch j on the current position pos_2 $C_j^1 > t_2$, then set $t_2 = C_j^1$ and go to the previous step to schedule batch j on the second stage.

5 Computational Experiments

The computational experiments aimed to compare the Lagrangian Heuristic and an integer program run on CPLEX software. The computational experiments were conducted by the second author on a personal computer with Intel Core *i5* processor *CPU@1.70* GHz, using Ubuntu 14.04 LTS, with base memory 4096 MB. The algorithms were implemented using C programming language. The test instances were generated randomly, with processing times chosen from the interval $[1, 10]$, weights chosen from the interval $(0, 2]$ and buffer requirements chosen from the interval $[100, 1000]$. A due date for each batch k was chosen from the interval $[\rho_k + 10, 2(\rho_k + 10)]$. Each test set consisted of ten instances which were defined by the number of jobs (25, 50 or 100), the number of batches (3, 5 or 10) and the number of machines on the first stage (2, 5 or 10). For each set the experiments were conducted for buffer sizes $B1 = b_{max}$, $B2 = 2b_{max}$ and $B3 = 3b_{max}$, where b_{max} is the maximum buffer requirement among all batches of an instance. Such randomly generated instances allowed to analyse the impact of variation of the parameters. The upper bound (28) was used for the planning horizon for both Lagrangian Heuristic and integer program run on CPLEX. The subgradient algorithm in the Lagrangian Heuristic was run for 1000 iterations, and the time limit for both - the heuristic and the integer program was 30 min for instances with 50 or 100 jobs, and 15 min for instances with 25 jobs. The Table 1 provides the summary of computational experiments. Each test set is represented in the first column in the format $n_b - N - M$, where for each instance of the set n_b is number of batches, N is number of jobs and M is number of machines on the first stage. Columns 2–4 contain the number of instances from each set, where the best solution *LH* found by the Lagrangian Heuristic is not worse than the best solution *IP* found by the integer program; columns 5–7 contain the number of instances from each set, where the run time of Lagrangian Heuristic t_{LH} is less than 50% of run time t_{IP} of the integer program executed on CPLEX; columns 8–10 contain the number of instances from each set, where the integer program found a feasible solution within the given time; columns 11–13 contain the number of instances from each set, where the integer program found an optimal solution.

The computational experiments have shown promising results and provided insights into how various combinations of buffer size, number of jobs and batches and number of machines on the first stage affect computational complexity. For large instances with 100 jobs the Lagrangian Heuristic provided a feasible solution within 2–15 min for most instances, with a few instances within 22–30 min; while direct integer program failed to obtain a feasible solution for many large instances within 30 min. However, for some large instance the best solution provided by Lagrangian Heuristic was achieved on the first iteration and it was not improved during the run of the algorithm. For medium instances with 50 jobs the Lagrangian Heuristic found better solutions for 30%–100% of instances for various sets depending on the number of machines on the first stage and buffer size. For small instances of 25 jobs the Lagrangian Heuristic provided not worse solutions than the integer program for some instances with smaller buffer $B1$ in

Table 1. Summary

Set	$LH \leq IP$			$t_{LH} \leq 0.5t_{IP}$			IP found feas			IP found opt		
	B1	B2	B3	B1	B2	B3	B1	B2	B3	B1	B2	B3
10-100-2	10	10	10	3	3	3	0	0	0	0	0	0
10-100-10	10	9	0	10	10	10	0	2	10	0	1	3
5-100-2	10	10	9	10	5	5	0	3	5	0	0	0
5-100-5	10	9	7	10	10	10	1	6	10	0	0	0
5-100-10	10	4	1	10	10	10	4	10	10	0	2	2
3-100-2	10	6	4	6	6	6	0	8	10	0	0	0
3-50-2	10	10	8	10	10	10	9	10	10	0	2	4
3-50-5	7	6	3	10	9	7	10	10	10	5	9	10
3-50-10	8	3	3	8	3	1	10	10	10	10	10	10
3-25-2	5	7	5	10	6	4	10	10	10	5	10	10
3-25-5	7	3	2	6	1	0	10	10	10	10	10	10

shorter time of 10–40 s, while it took up to 15 min for the integer program to find a feasible solution. However for larger buffer sizes $B2$ and $B3$ the integer program solved all small instances to optimality and outperformed the in terms of time and quality of solutions for most of the instances. It would be interesting to compare the solution obtained by the Lagrangian Heuristic with the optimum one for all instances. However it took 7 h for CPLEX to improve by 2.8% a feasible (still not optimal) solution for the instance with 100 jobs partitioned into 10 batches, 10 parallel machines on the first stage and the buffer capacity $B3$. For the similar instance with 2 parallel machines and a smaller buffer $B1$, in 7 h CPLEX could not even find a feasible solution.

6 Conclusion

This paper is concerned with the two-stage flexible flowshop with batch processing and a limited buffer. The buffer requirements vary from batch to batch. The objective function is the total weighted tardiness. This problem is new and is applicable to a range of areas from supply chains to multimedia computerised systems. For this NP -hard problem, the paper presents an integer program and a new Lagrangian relaxation based algorithm that decomposes the problem with efficient algorithm for each part of this decomposition. The computational experiments demonstrated superiority of the presented approach to the straightforward application of CPLEX optimisation software.

References

1. Brucker, P., Knust, S.: Complex Scheduling. GOR-Publications. Springer, Heidelberg (2012). <https://doi.org/10.1007/978-3-642-23929-8>
2. Emmons, H., Vairaktarakis, G.: Flow Shop Scheduling: Theoretical Results, Algorithms, and Applications. Springer, New York (2013). <https://doi.org/10.1007/978-1-4614-5152-5>
3. Fisher, M.L.: The Lagrangian relaxation method for solving integer programming problems. *Manage. Sci.* **50**(12), 1861–1871 (2004)
4. Fung, J., Zinder, Y.: Permutation schedules for a two-machine flow shop with storage. *Oper. Res. Lett.* **44**(2), 153–157 (2015)
5. Fung, J., Singh, G., Zinder, Y.: Capacity planning in supply chains of mineral resources. *Inf. Sci.* **316**, 397–418 (2015)
6. Garey, M., Johnson, D., Sethi, R.: The complexity of flowshop and jobshop scheduling. *Math. Oper. Res.* **1**(2), 117–129 (1976)
7. Gu, H., Memar, J., Zinder, Y.: Efficient Lagrangian heuristics for the two-stage flow shop with job dependent buffer requirements. In: IWOCA 2017 Proceedings. LNCS (2017, accepted)
8. Kononov, A., Hong, J.-S., Kononova, P., Lin, F.-C.: Quantity-based buffer-constrained two-machine flowshop problem: active and passive prefetch models for multimedia applications. *J. Sched.* **15**(4), 487–497 (2012)
9. Kononova, P.A., Kochetov, Y.A.: The variable neighbourhood search for two machine flowshop problem with passive prefetch. *J. Appl. Ind. Math.* **19**(5), 63–82 (2013)
10. Lin, F.-C., Hong, J.-S., Lin, B.M.T.: A two-machine flowshop problem with processing time-dependent buffer constraints - an application in multimedia presentations. *Comput. Oper. Res.* **36**(4), 1158–1175 (2009)
11. Lin, F.-C., Hong, J.-S., Lin, B.M.T.: Sequence optimization for media objects with due date constraints in multimedia presentations from digital libraries. *Inf. Syst.* **38**(1), 82–96 (2013)
12. Lin, F.-C., Lai, C.-Y., Hong, J.-S.: Minimize presentation lag by sequencing media objects for auto-assembled presentations from digital libraries. *Data Knowl. Eng.* **66**(3), 382–401 (2008)
13. Lin, F.-C., Lai, C.-Y., Hong, J.-S.: Heuristic algorithms for ordering media objects to reduce presentation lags in auto-assembled multimedia presentations from digital libraries. *Electron. Libr.* **27**(1), 134–148 (2009)
14. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems. Springer, New York (2012). <https://doi.org/10.1007/978-1-4614-2361-4>
15. Witt, A., Voß, S.: Simple heuristics for scheduling with limited intermediate storage. *Comput. Oper. Res.* **34**(8), 2293–2309 (2007)

Computing Periods...

Junhee Cho^(✉), Sewon Park, and Martin Ziegler

KAIST School of Computing, Daejeon, Republic of Korea
{junheecho,sewon,ziegler}@kaist.ac.kr

Abstract. A *period* is the difference between the volumes of two semi-algebraic sets. Recent research has located their worst-case complexity in low levels of the Grzegorzcyk Hierarchy. The present work introduces, analyzes, and evaluates three rigorous algorithms for rigorously computing periods: a deterministic, a randomized, and a ‘transcendental’ one.

Keywords: Exact Real Computation · Reliable numerics
Computational algebraic geometry · Randomized algorithms

1 Introduction

A *period* is the absolutely convergent integral of a multivariate rational function with integer coefficients over Euclidean domains given by polynomial inequalities with integer coefficients [KZ01]:

$$\int_{\Delta} \frac{p(x_1, \dots, x_d)}{q(x_1, \dots, x_d)} dx_1 \cdots dx_d, \quad (1)$$

where $\Delta \subseteq \mathbb{R}^d$ is a Boolean combination of strict and non-strict polynomial inequalities $p_j(\vec{x}) > 0$ and $q_i(\vec{x}) \geq 0$ over, like p and q , integer coefficients: $p, q, p_1, \dots, p_J, q_1, \dots, q_I \in \mathbb{Z}[X_1, \dots, X_d]$. Periods are receiving increasing interest in Algebraic Model Theory as they have finite descriptions (the polynomials’ coefficients) and include all algebraic reals as well as some transcendentals:

$$\sqrt{2} = \int_{t:2t^2 \leq 1} t dt, \quad \ln(x) = \int_1^x 1/t dt = \int_{t \leq x, s \cdot t \leq 1} 1 ds dt, \quad \pi = \int_{x^2 + y^2 \leq 1} 1 dx dy \quad (2)$$

Every period can be expressed as difference of two semi-algebraic volumes: For co-prime $p, q \in \mathbb{Z}[X_1, \dots, X_d]$, Eq. (1) translates to

$$\int_{\Delta_{p,q,+}} 1 d\vec{x} dy - \int_{\Delta_{p,q,-}} 1 d\vec{x} dy = \text{vol}(\Delta_{p,q,+}) - \text{vol}(\Delta_{p,q,-}), \quad (3)$$

Based on ideas presented at CCA 2017, this work was supported by the National Research Foundation of Korea (grant NRF-2017R1E1A1A03071032) and the International Research & Development Program of the Korean Ministry of Science and ICT (grant NRF-2016K1A3A7A03950702). We thank the anonymous referees for feedback!

where $\Delta_{p,q,+} := \{(\vec{x}, y) : 0 \leq y \cdot q(\vec{x}) \leq p(\vec{x}) \wedge q(\vec{x}) > 0\} \cup \{(\vec{x}, y) : 0 \geq y \cdot q(\vec{x}) \geq p(\vec{x}) \wedge q(\vec{x}) < 0\}$ and $\Delta_{p,q,-} := \{(\vec{x}, y) : 0 \geq y \cdot q(\vec{x}) \geq p(\vec{x}) \wedge q(\vec{x}) > 0\} \cup \{(\vec{x}, y) : 0 \leq y \cdot q(\vec{x}) \leq p(\vec{x}) \wedge q(\vec{x}) < 0\}$. Note that $\Delta_{p,q,+}, \Delta_{p,q,-} \subseteq \mathbb{R}^{d+1}$ may be unbounded even when Δ was bounded. However by means of Singularity Resolution one can w.l.o.g. restrict to compact domains [VS17, Theorem 1.1]. This shows that sum and (Cartesian) product of periods are again periods.

Many interesting open questions evolve around periods; for instance [KZ01, Problem 1] of whether there exists an algorithm that, given two representations of periods, decides whether they are equal or not? Or [KZ01, Problem 3] asking for an ‘explicit’ example of a real number that is not a period. Inner-mathematical candidates are $1/\pi$ and $e = \sum_n 1/n!$, but proving so seems infeasible with the current methods. The family of (semi-algebraic domains and thus also of) periods being countable, non-periods must be abundant.

In fact every period is computable in the sense of Recursive Analysis [Tur37, Wei00]; therefore any uncomputable real, such as the Halting Problem encoded in binary or random reals [BDC01] like Chaitin’s Ω , cannot be periods. More precisely each period is of *lower elementary* complexity in Grzegorzczuk’s Hierarchy [Yos08, TZ10, SWG12]. Note that such improved upper complexity bounds give rise to more candidates of non-periods.

Problem 1. Characterize the computational complexity of periods!

Moreover efficient algorithms for computing (i.e. producing guaranteed high-precision approximations to) periods enable Experimental Mathematics [KZ01, Problem 2]; cmp. [Bai17].

We present three such algorithms: a deterministic one, a randomized (Las Vegas) one, and a transcendental one (to be clarified below). We prove them correct; describe their implementation in the convenient *Exact Real Computation* paradigm; estimate their cost in the (possibly unrealistic) unit cost model; and empirically analyze and compare their behaviour in terms of the output precision and the degree of the polynomial involved.

Subsection 1.1 recalls central notions, properties, and practice of real computation; Subsect. 1.2 puts things in perspective to related work. Our algorithms are presented, and proven correct, in Sect. 2. In Sect. 3 we introduce our implementation, performance measurements, and their evaluation/interpretation. Section 4 expands on future work.

1.1 Real Computation

Regular floating-point arithmetic incurs rounding errors that accumulate over time and hamper reliable computations. Interval calculations keep track of the error bounds—which may blow up beyond use and due to overlap render comparisons meaningless. The present work peruses the `iRRAM C++` library [Mül01, MZ14], providing real numbers as abstract data type with exact operations and partial comparison: A test “ $y > 0$ ” freezes in case $y = 0$.

Indeed it is well-known from Recursive Analysis that equality of real (and not just algebraic) numbers is equivalent to the complement of the Halting Problem [Wei00, Exercise 4.2.9]. Here, computing $y \in \mathbb{R}$ means to produce dyadic approximations $a_n/2^n$, $a_n \in \mathbb{Z}$, to y up to absolute error $1/2^n$; similarly for real arguments x .

To write total programs in spite of comparison being partial, a *parallel disjunction* is provided: calling the non-deterministic or *multivalued* ‘function’ $\text{choose}(x_1 > 0, \dots, x_k > 0)$ returns *some* integer j such that $x_j > 0$ holds, provided such j exists.

Subject to this modified semantics of tests, *Exact Real Computation* allows to conveniently process real arguments and intermediate results as entities, naïvely without precision considerations. On the other hand the output/return value of a function in **iRRAM** merely needs to be provided in approximation up to absolute error 2^p for any parameter $p \in \mathbb{Z}$ passed. The following algorithm demonstrates this paradigm with the *trisection* method for finding the (promised unique and simple) root of a given continuous function $f : [0; 1] \rightarrow [-1; 1]$ while avoiding the sign test “ $0 < f(a) \cdot f(b')$ ” to fail in case b' already happens to be a root:

Algorithm . REAL Trisection(INTEGER p , REAL \rightarrow REAL f)

```

1: REAL  $\ni a := 0$ ; REAL  $\ni b := 1$ 
2: while  $\text{choose}(b - a > 2^{p-1}, 2^p > b - a) == 1$  do
3:   REAL  $a' := \frac{2}{3}a + \frac{1}{3}b$ ;   REAL  $b' := \frac{1}{3}a + \frac{2}{3}b$ ;
4:   if  $\text{choose}(0 > f(a) \cdot f(b'), 0 > f(a') \cdot f(b)) == 1$ 
5:     then  $b := b'$  else  $a := a'$  end if
6: end while;   return  $a$ 

```

Remark 2. A caveat, in *Exact Real Computation* the bit—as opposed to unit—cost of each operation may well depend on the value (and internal precision) of the data being processed. For instance a sign test “ $x > 0$ ” will take time between linear and quadratic in $n \approx \log_2(1/|x|)$: to obtain the dyadic approximation $a_n/2^n$ of x up to error 2^{-n} and verify it to be strictly larger than 2^{-n} . Similarly, a parallel test $\text{choose}(x_1 > 0, \dots, x_k > 0)$ will take time roughly $k \cdot \min_{j < k} \log(1/|x_j|)$.

1.2 Periods and Their Computational Complexity

It has been shown that periods are of *lower elementary* complexity [TZ10, SWG12]. The present subsection recalls this and related notions with their connections to resource-bounded complexity.

In Grzegorzcyk’s Hierarchy, *Lower Elementary* means the smallest class of total multivariate functions $f : \mathbb{N}^d \rightarrow \mathbb{N} = \{0, 1, 2, \dots\}$ containing the constants, projections, successor, modified difference $x \dot{-} y = \max\{x - y, 0\}$, and is closed under composition and bounded summation $f(\vec{x}, y) = \sum_{z=0}^y g(\vec{x}, z)$.

We write $\mathcal{M}^2 = \mathcal{E}^2$ for the smallest class of such f containing the constants, projections, successor, modified difference, binary multiplication, and

closed under both composition and bounded search $\mu(f)(\vec{x}, y) = \min\{z \leq y : f(\vec{z}, z) = 0\}$.

A real number r is *lower elementary* if there exist lower elementary integer functions $f, g, h : \mathbb{N} \rightarrow \mathbb{N}$ with $|r - \frac{f(N)-g(N)}{h(N)}| < 1/N$ for all $N > 0$; similarly for a real number in \mathcal{M}^2 .

A real number r is computable in *time* $t(n)$ and *space* $s(n)$ if a Turing machine can, given $n \in \mathbb{N}$ and within these resource bounds, produce some $a_n \in \mathbb{Z}$ with $|r - a_n/2^n| \leq 2^{-n}$ [Ko91].

Fact 3

- (a) All functions from \mathcal{M}^2 are lower elementary; and the latter functions grow at most polynomially in the value of the arguments. In terms of the binary input length and with respect to bit-cost, lower elementary functions are computable using a linear amount of memory for intermediate calculations and output, that is, they belong to the complexity class $\text{FSPACE}(n)$.
- (b) $\text{FSPACE}(n)$ is closed under bounded summation and therefore coincides with the class of lower elementary functions. The 0/1-valued functions (that is, decision problems) in \mathcal{M}^2 exhaust the class $\text{SPACE}(n)$ [Rit63, Sect. 4]; cmp. [Kut87].
- (c) π and $e = \sum_n 1/n!$ and Liouville's transcendental number $L = \sum_n 10^{-n!}$ and the Euler-Mascheroni Constant $\gamma = \lim_n (-\ln(n) + \sum_{k=1}^n 1/k)$ are all lower elementary [Sko08, Sect. 3].
- (d) The set of lower elementary real numbers constitutes a real closed field: Binary sum and product and reciprocal of lower elementary real numbers, as well as any real root of a non-zero polynomial with lower elementary coefficients, are again lower elementary [SWG12, Theorem 2].
- (e) Arctan, natural logarithm and exponential as well as Γ and ζ function map lower elementary reals to lower elementary reals [TZ10, Sect. 9].
- (f) Natural logarithm maps periods to periods; $\zeta(s)$ is a period for every integer $s \geq 2$ [KZ01, Sect. 1.1].
- (g) Periods are lower elementary [TZ10, Corollary 6.4].
- (h) Given a Boolean expression $\varphi(x_1, \dots, x_m)$ as well as the degrees and coefficients of the polynomials p_j defining its constituents S_{p_j} , deciding whether the semi-algebraic set $\varphi(S_{p_1}, \dots, S_{p_m})$ is non-empty/of given dimension [Koi99] is complete for the complexity class $\text{NP}_{\mathbb{R}}^0 \supseteq \text{NP}$.

Item (a) follows by structural induction. Together with (b) it relates resource-oriented to Grzegorzcyk's structural Complexity Theory. Note that the hardness Result (h) does not seem to entail a lower bound on the problem of approximating a fixed volume; in fact many of the usual reductions among real algebraic decision problems [Mee06] fail under volume considerations. Common efficient and practical algorithms tailored for approximating L , e , γ , or the period π do so up to absolute error $1/N := 2^{-n}$ within time polynomial in the binary precision parameter $n = \log_2 N$ [Kan03]; whereas the best runtime bound known for $\text{SPACE}(n)$ is only exponential [Pap94, Problem 7.4.7]. On the other hand exponential-time algorithms may well be practical [FG06, KF10].

2 Our Algorithms and Their Analyses

In view of Eq. (3) and Subjects. 1.1 and 1.2, this section devises and analyzes algorithms that approximate, up to guaranteed absolute error 2^{-n} , the volume of the set of solutions \vec{x} to a (given) disjunction of m conjunctions of monic polynomial inequalities, each of maximum degree $\leq k$ in d variables with integer coefficients between -2^ℓ and $+2^\ell$.

By appropriate integer scaling and shifting, it suffices to restrict to the unit cube $[0; 1]^d$ and to strict inequalities; see Fact 5(a) below. The common basic idea underlying all of our algorithms is to divide $[0; 1]^d$ into sub-cubes

$$Q_{\vec{c}, N} := \left[\frac{\vec{c}}{N}; \frac{\vec{c} + \vec{1}}{N} \right) = \prod_{i=1}^d \left[\frac{c_i}{N}; \frac{c_i + 1}{N} \right), \quad \vec{c} \in [N]^d, \quad (4)$$

where $[N] := \{0, \dots, N - 1\}$ and $\vec{1} := (1, \dots, 1)$; then determine the signs of the polynomials p_j in *some* point $\vec{x}_{\vec{c}, N} \in Q_{\vec{c}, N}$; and count those, where the constraints $p_j(\vec{x}_{\vec{c}, N}) > 0$ are met, with uniform weight $\text{vol}(Q_{\vec{c}, N}) = N^{-d}$.

However (I) a polynomial's sign may vary within a $Q_{\vec{c}, N}$, the above approach can incur an error. Moreover (II) $\vec{x}_{\vec{c}, N}$ may happen to be (close to) a root of p_j ; in which case determining the sign of $p_j(\vec{x}_{\vec{c}, N})$ may take long in terms of bit-cost, or fail entirely. The sequel describes our approaches to still achieve totally correct algorithms: Subject. 2.1 takes care of (I), while Subjects. 2.3, 2.2, and 2.4 describe three different approaches to avoid (II).

2.1 Recap on Real Algebraic Geometry

Regarding (I) in the case $d = 1$ a sign change can affect, namely occur in, at most $m \cdot k$ of the sub-intervals $Q_{c, N}$: because each of the m univariate polynomials of degree $\leq k$ can have at most k roots. So taking $N \geq m \cdot k \cdot 2^n$ guarantees the required error bound 2^{-n} . A multivariate polynomial on the other hand may have infinitely many roots—which however can form only a bounded number of connected components. This allows us to generalize the 1D analysis of (I) as follows:

Lemma 4

- (a) In case $d = 2$ and for any fixed non-zero polynomial $p \in \mathbb{R}[X, Y]$ of maximum degree k , at most $1 + (k - 1) \cdot (k - 2)/2 + 2(N + 1) \cdot k$ of the $N \times N$ sub-squares $Q_{\vec{c}, N}$ can contain roots of p .
- (b) In case $d = 2$ and for an arbitrary finite family of non-zero polynomials p_j of maximum degree k , at most $1 + (2k - 1) \cdot (2k - 2)/2 + 4(N + 1) \cdot k$ of the $N \times N$ sub-squares $Q_{\vec{c}, N}$ can contain simultaneous roots of all the p_j .
- (c) For $d \geq 3$ and any fixed non-zero polynomial $p \in \mathbb{R}[X_1, \dots, X_d]$ of maximum degree k , at most $k^d + k^{d-1} \cdot d \cdot (N + 1)$ of the N^d sub-(hyper)cubes $Q_{\vec{c}, N}$ can contain roots of p .
- (d) For $d \geq 3$ and an arbitrary finite family of non-zero polynomials of maximum degree k , at most $k \cdot (2k - 1)^d + k \cdot (2k - 1)^{d-2} \cdot d \cdot (N + 1)$ of the N^d sub-(hyper)cubes $Q_{\vec{c}, N}$ can contain simultaneous roots of all of them.

(e) It holds $k^d + k^{d-1} \cdot d \cdot (N + 1) \leq N^d \cdot 2^{-n}$ for all $N \geq 3k \cdot 2^{n/(d-1)}$ and $k \geq d \geq 3$.

Note that the number m of polynomials in a conjunction $\bigcap_{j=1}^m p_j(\vec{x}) = 0$ barely affects the above bounds, since in the real setting it is equivalent to the single equation of double degree $\sum_{j=1}^m p_j^2(\vec{x}) = 0$.

In order to guarantee absolute error bound 2^{-n} , all our algorithms described in the following subsections will in case $d = 2$, rather than apply the concise but asymptotic Lemma 4(e), build on Lemma 4(a) and use binary search to find the least $N \in \mathbb{N}$ satisfying $N^2 \geq 2^n \cdot (1 + (k - 1) \cdot (k - 2)/2 + 2(N + 1) \cdot k)$.

Proof

- (a) By Fact 5(c) below, the roots of p can form at most $c \leq 1 + (k - 1) \cdot (k - 2)/2$ connected components in \mathbb{R}^2 . Of course such a component may extend through more than one of the sub-squares $Q_{\vec{c},N}$; however in order to do so, it must cross one of the $N + 1$ horizontal lines or one of the $N + 1$ vertical lines forming the sub-division of $[0; 1]^2$. More precisely for component C to extend to $M_C \in \mathbb{N}$ of the N^2 sub-squares, it must intersect at least $M_C - 1$ of the $2(N + 1)^2$ segments of the $2(N + 1)$ aforementioned lines; and for all c components to extend to a total of M of the N^2 sub-squares, they have to intersect these lines in at least $M - c$ points—distinct points, since connected components do not meet. However p restricted to any of the $2(N + 1)$ lines boils down to a univariate polynomial (either in X or in Y) of degree at most k , and hence can have at most k roots on each such line: requiring $M - c \leq 2k \cdot (N + 1)$.
- (b) Joint roots of the real p_j of maximum degree k are precisely the roots of the single polynomial $\sum_j p_j^2$ of maximum degree $2k$.
- (c) Similarly to the proof of (a), the roots of p can form at most $c_d \leq k^d$ connected components according to Fact 5(d). For any such component C to extend to $M_C \in \mathbb{N}$ of the N^d sub-(hyper)cubes, it must intersect at least $M_C - 1$ of the $d \cdot (N + 1)^d$ facets of the overall subdivision induced by the $d \cdot (N + 1)$ hyperplanes; and for all c_d components to extend to a total of M of the N^d sub-cubes, they have to intersect these hyperplanes in at least $M - c_d$ different components! However p restricted to any of the $d \cdot (N + 1)$ hyperplanes boils down to a $(d - 1)$ -variate polynomial of maximum degree at most k , whose roots can form at most $c_{d-1} \leq k^{d-1}$ connected components according to Fact 5(d): $M - c_d \leq c_{d-1} \cdot d \cdot (N + 1)$.
- (d) Similarly.
- (e) Apply inequality $|x|^d + |y|^d \leq (|x| + |y|)^d$ to $x^d := 2^n \cdot (k^d + d \cdot k^{d-1}) \leq 2k^d \cdot 2^{n \cdot d/(d-1)}$ and $y^d := N \cdot d \cdot k^{d-1} \cdot 2^n$, taking into account $\sqrt[d]{2} + \sqrt[d-1]{d} \leq 3$ for all $d \geq 3$. \square

Fact 5

- (a) The set $\{\vec{x} : p(\vec{x}) = 0\}$ of roots of a non-zero polynomial p has measure zero.
- (b) Let $S_1, \dots, S_d \subseteq \mathbb{R}$ be arbitrary subsets of cardinality $|S_i| > k$ and $p \in \mathbb{R}[X_1, \dots, X_d]$ non-zero of maximal degree $\leq k$. Then there exists some $\vec{x} \in \prod_{i=1}^d S_i$ with $p(\vec{x}) \neq 0$.

- (c) Let $p \in \mathbb{R}[X, Y]$ denote a bivariate polynomial of maximum degree k . Then the number of connected components of $\{(x, y) : x, y \in \mathbb{R}, p(x, y) = 0\}$ is at most $1 + (k - 1) \cdot (k - 2)/2$.
- (d) If $\Delta \subseteq \mathbb{R}^d$ is the zero set of one polynomial of maximum degree k , then it has at most k^d connected components; if it is the conjunction of (any number of) such sets, then it has at most $k \cdot (2k - 1)^d$ connected components.
- (e) For p_1, \dots, p_d pairwise distinct primes, $\vec{x} := (e^{\sqrt{2}}, e^{\sqrt{3}}, \dots, e^{\sqrt{p_d}})$ is algebraically independent: $q(\vec{x}) \neq 0$ for every non-zero d -variate polynomial q with integer coefficients; and more generally $q(A \cdot \vec{x} + \vec{b}) \neq 0$ for any vector $\vec{b} \in \mathbb{A}^d$ with algebraic coefficients and invertible $d \times d$ -matrix $A \in \text{GL}_d(\mathbb{A})$.

Claim (b) strengthens (a) and proceeds by induction on d : For any fixed $(x_1, \dots, x_{d-1}) \in \prod_{i=1}^{d-1} S_i$, $p(x_1, \dots, x_{d-1}, X_d)$ is a univariate polynomial of degree $\leq k$. Claim (c) is *Harnack’s Curve Theorem* [PP16, Theorem 48.1], (d) its generalization due to Milnor and Thom [HRR90, Theorem 9]; (e) is the *Lindemann-Weierstrass Theorem*, applied to linear independence of prime square roots over rationals.

2.2 A Real Randomized Algorithm

In order to avoid (II) accidentally hitting a root \vec{x} of some constraint polynomial p when trying to determine its sign in the sub-cube $Q_{\vec{c}, N} \ni \vec{x}$, randomization provides an arguably easiest solution: By Fact 5 the probability for this to occur is zero. So picking a random $\vec{x} \in Q_{\vec{c}, N}$ gives rise to a Las Vegas-type algorithm: always correct, but with running time proportional to $\log(1/p(\vec{x}))$; recall Remark 2.

Randomization has become ubiquitous in Theoretical Computer Science—regarding discrete problems [MU05]: In the real setting it has only (yet thoroughly) been considered with respect to computability; cmp. [BGH15].

Here we have designed the first truly real random number generator in *Exact Real Computation*. Each call produces some $r \in [0; 1]$ independently with respect to the uniform distribution: Repeating d times, shifting by \vec{c} and scaling with $1/N$ then yields the sought $\vec{x} \in Q_{\vec{c}, N}$. Internally our real generator in turn builds on a generic source of independent fair coin flips; equivalently: independent uniformly distributed integers in the range from 0 to $2^n - 1$, for any given $n \in \mathbb{N}$. (The uniform distribution on $[0; 1]$ is then easily converted to other popular continuous ones such as, say, Gaussian on \mathbb{R} .)

2.3 A Deterministic Algorithm

With the dimension d fixed, we can avoid the problem of derandomizing *Polynomial Identity Testing* and still get an efficient deterministic algorithm in *Exact Real Computation* for finding the sign of any given non-zero $p \in \mathbb{R}[X_1, \dots, X_d]$ of maximal degree $\leq k$ in *some* non-root $\vec{x} \in Q_{\vec{c}, N}$: Based on Fact 5(b), evaluate p ’s sign on the $(k + 1)^d$ points of a $(k + 1) \times \dots \times (k + 1)$ grid in $Q_{\vec{c}, N}$ in parallel, recall Remark 2.

2.4 A Transcendental Algorithm

By Fact 5(e), algebraically independent arguments $\vec{x} = (x_1, \dots, x_d)$ avoid problem (II) and can be computed efficiently: deterministically. (Any tuple of independent *random* reals of course is algebraically independent with certainty.) This algorithm thus takes such a \vec{x} and scales and shifts it by dyadic rationals to lie in $Q_{\vec{c}, N}$.

3 Implementation and Evaluation

Evaluating a given d -variate polynomial of maximal degree $\leq k$ takes $\mathcal{O}(k^d)$ arithmetic operations. Combined with Lemma 4(e), we conclude that the number of operations performed by the randomized and by the transcendental algorithm (Subsects. 2.2 and 2.4) to achieve guaranteed output absolute error 2^{-n} is $\mathcal{O}(k^{d+1} \cdot 2^{n/(d-1)})$; the deterministic algorithm (Subsect. 2.3) incurs an additional factor $\mathcal{O}(k^d)$.

However this analysis only counts the number of operations, that is, referring to an algebraic or unit-cost measure—as opposed to the more realistic bit-cost measure taking into account aspects of internal or working precision.

The latter seem hard to estimate, though, since they depend not only on the output precision n and the polynomial degree k , but also on (the coefficient vector of) the polynomial constraints p under consideration: which in the worst-case may give rise to unbounded running times. For a more realistic assessment we have thus implemented, empirically evaluated and compared the practical performance of the above three algorithms on the following kinds of benchmark polynomials:

The bivariate polynomials $X^2 + Y^2 - 1$ and $(X - 2)^2 + (X \cdot Y - 1)^2$ representing¹ the transcendental periods π and $\ln(2)$; recall Eq. (2); and for $d = 2$ and $d = 3$ the multivariate scaled *Wilkinson-type* Polynomials I $p_{n,d} := \prod_{i=1}^d \prod_{j=1}^k (X_i - \frac{j}{k})$ deliberately placing roots at points on a grid that the deterministic algorithm will try to determine their signs in and *Wilkinson-type* Polynomials II $p_{n,d} := \prod_{j=1}^k (\prod_{i=1}^d X_i - \frac{j}{k})$.

3.1 Computing Environment

The above three algorithms were implemented without multithreading in *Exact Real Computation* based on the iRRAM C++ library [Mül01, MZ14] commit 487a123. Their source codes and the experiment results are available for download from url <https://github.com/junheecho/period> and <https://github.com/junheecho/iRRAM>. We have executed and timed them on a computer with Intel® Core™ i7-6950X processor with 10 cores, 20 threads running at 3.00 GHz and 64 GB of RAM. Less than 20 processes ran at the same time so that they do not impede each other. The source code is compiled with g++ 5.4.0 on Ubuntu 16.04.3 LTS. We focus on CPU time; memory was never a problem.

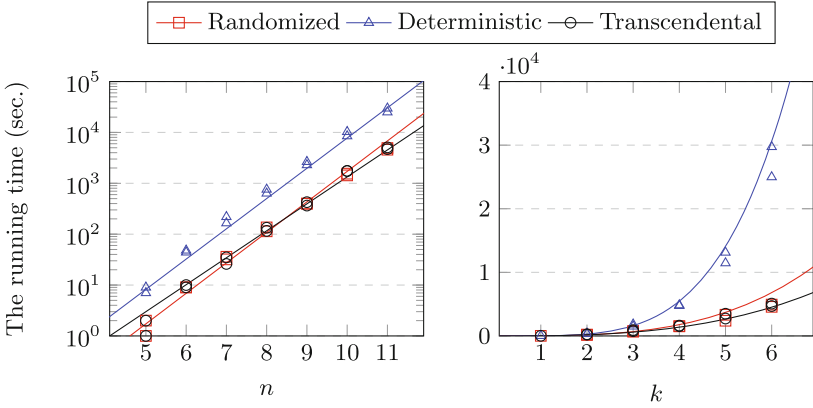
¹ Of course the specific periods π and $\ln(2)$ admit other, more efficient algorithms.

3.2 Performance Results

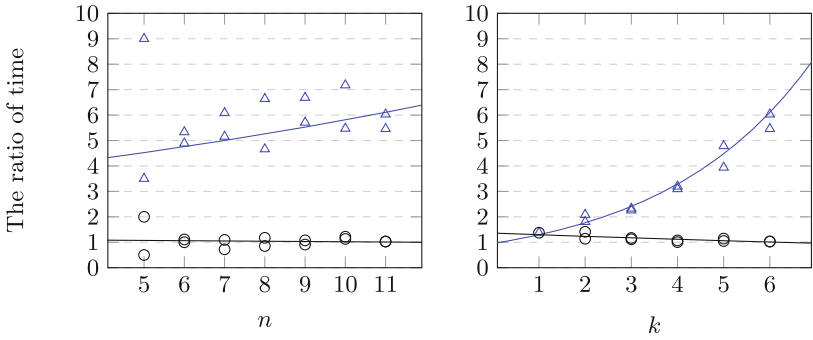
We have measured, for each of the three above algorithms and each of the above benchmark polynomials as input, the CPU time in dependence on n ; for the Wilkinson-type Polynomials also on k . We have then fitted the results to the model/ansatz $\exp(n \cdot \beta - \alpha)$ —for the Wilkinson-type Polynomials to $\exp(n \cdot \beta - \alpha) \cdot k^\gamma$ —and plotted both. We have also fitted the result for the Wilkinson-type Polynomials to $\exp(k \cdot \gamma + n \cdot \beta - \alpha)$ but the R^2 scores show the aforementioned model is more suitable. We have also plotted and fitted the *ratios* of the algorithms’ respective performances to the aforementioned models: See the following figures (Table 1 and Fig. 1).

Table 1. Parameter regression of CPU time

Input	Algorithm	Regression	R^2 score
π	Randomized	$\exp(1.31n - 9.51)$	0.99
	Deterministic	$\exp(1.14n - 7.69)$	0.88
	Transcendental	$\exp(1.34n - 9.73)$	1.00
(the ratio over the randomized)	Deterministic	$\exp(-3 \cdot 10^{-5}n + 0.03)$	0.00
	Transcendental	$\exp(-0.07n + 0.88)$	-0.66
$\ln(2)$	Randomized	$\exp(1.38n - 11.74)$	1.00
	Deterministic	$\exp(1.30n - 10.65)$	1.00
	Transcendental	$\exp(1.31n - 10.75)$	1.00
(the ratio over the randomized)	Deterministic	$\exp(-0.09n + 1.09)$	0.62
	Transcendental	$\exp(0.07n - 0.57)$	0.38
Wilkinson-type Polynomials I	Randomized	$\exp(1.38n - 12.32) \cdot k^{3.33}$	0.89
		$\exp(1.01k + 1.37n - 11.95)$	0.49
	Deterministic	$\exp(1.37n - 12.34) \cdot k^{4.24}$	0.98
		$\exp(1.30k + 1.37n - 12.10)$	0.51
	Transcendental	$\exp(1.22n - 10.27) \cdot k^{2.94}$	0.95
		$\exp(0.90k + 1.21n - 10.01)$	0.86
(the ratio over the randomized)	Deterministic	$\exp(0.05n - 0.54) \cdot k^{0.94}$	0.63
		$\exp(0.31k + 0.05n - 0.60)$	0.73
	Transcendental	$\exp(-0.01n + 0.41) \cdot k^{-0.17}$	-0.05
		$\exp(-0.05k - 0.01n + 0.42)$	-0.06
Wilkinson-type Polynomials II	Randomized	$\exp(1.25n - 10.39) \cdot k^{2.80}$	0.98
		$\exp(0.73k + 1.24n - 9.82)$	0.76
	Deterministic	$\exp(1.27n - 10.47) \cdot k^{3.55}$	0.99
		$\exp(0.95k + 1.28n - 10.01)$	0.88
	Transcendental	$\exp(1.26n - 10.19) \cdot k^{2.64}$	0.99
		$\exp(0.66k + 1.24n - 9.36)$	0.92
(the ratio over the randomized)	Deterministic	$\exp(0.08n - 1.03) \cdot k^{1.03}$	0.73
		$\exp(0.28k + 0.08n - 0.96)$	0.85
	Transcendental	$\exp(0.02n + 0.12) \cdot k^{-0.15}$	0.11
		$\exp(-0.04k + 0.02n + 0.07)$	0.12



(a) The running time and the fit, $k = 6$ (left) and $n = 11$ (right)



(b) The ratio over the randomized, $k = 6$ (left) and $n = 11$ (right)

Fig. 1. CPU time (sec.) of computing periods via Wilkinson-type Polynomials I

3.3 Interpretation

Our measurements confirm the predicted running times polynomial in k and exponential in n . They furthermore exhibit an exponential advantage of the randomized and the transcendental algorithm over the deterministic one. The performances of the randomized and the transcendental algorithms are identical. The difference of performance is little with respect to n , but significant with respect to k ; thus, the performances of all algorithms are identical when computing π and $\ln(2)$ because n is the only parameter.

Following the *Balls-into-Bins* paradigm [BCSV06], we have tried a synthesis of the randomized and the deterministic algorithm (Subsects. 2.2 and 2.3) that evaluates the polynomial's sign at *two* random points in parallel—however with no benefit in performance.

4 Conclusion and Perspectives

We have present three algorithms rigorously computing periods: a deterministic one, that evaluates the constraint polynomials at sufficiently many dyadic points simultaneously such as to guarantee at least one missing its roots; a randomized (and arguably first rigorous real) one, that misses roots almost surely; and one evaluating at an appropriate algebraically independent argument that by definition cannot constitute a root.

Although all three take time exponential in the output precision n , they exhibit significant differences in practical performance. Perhaps surprisingly, evaluating at two random points in parallel (and thus automatically choosing the ‘better’ one) turned out to be slower, not faster, than a single one.

For now we have focused on the case of two (and three) variables. Future work will extend in that, and the following directions:

- (a) Picking up on the first paragraph of Sect. 3, we will try to identify reasonable parameters of the polynomials $p \in \mathbb{Z}[X_1, \dots, X_d]$ under consideration, in addition to their maximal degree bound k , to devise a refined rigorous parameterized bit-cost analysis.
- (b) The question remains open as of whether every fixed period can be computed (i.e. approximated up to absolute error 2^{-n}) in time polynomial in n .
- (c) In the spirit of *Experimental Mathematics*, we plan to algorithmically search for new (candidate, linear or algebraic) relations among periods.

References

- [Bai17] Bailey, D.H.: Jonathan Borwein: experimental mathematician. *Exp. Math.* **26**(2), 125–129 (2017)
- [BCSV06] Berenbrink, P., Czumaj, A., Steger, A., Vöcking, B.: Balanced allocations: the heavily loaded case. *SIAM J. Comput.* **35**(6), 1350–1385 (2006)
- [BDC01] Becher, V., Daicz, S., Chaitin, G.: A highly random number. In: Calude, C.S., Dinneen, M.J., Sburlan, S. (eds.) *Combinatorics, Computability and Logic. Discrete Mathematics and Theoretical Computer Science*, pp. 55–68. Springer, London (2001). https://doi.org/10.1007/978-1-4471-0717-0_6
- [BGH15] Brattka, V., Gherardi, G., Hölzl, R.: Las Vegas computability and algorithmic randomness. In: Mayr, E.W., Ollinger, N. (eds.) *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015). Leibniz International Proceedings in Informatics (LIPIcs)*, vol. 30, pp. 130–142. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Dagstuhl (2015)
- [FG06] Flum, J., Grohe, M.: *Parameterized Complexity Theory. Texts in Theoretical Computer Science. An EATCS Series*. Springer, Heidelberg (2006). <https://doi.org/10.1007/3-540-29953-X>
- [HRR90] Heintz, J., Recio, T., Roy, M.-F.: Algorithms in real algebraic geometry and applications to computational geometry. In: Goodman, J.E., Pollack, R., Steiger, W. (eds.) *Discrete and Computational Geometry: Papers from the DIMACS Special Year. DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 6, pp. 137–164. DIMACS/AMS (1990)

- [Kan03] Kanada, Y.: 計算機による円周率計算 (特集 円周率 π). *J. Math. Cult.* **1**(1), 72–83 (2003)
- [KF10] Kratsch, D., Fomin, F.V.: *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (2010). <https://doi.org/10.1007/978-3-642-16533-7>
- [Ko91] Ko, K.-I.: *Complexity Theory of Real Functions*. Progress in Theoretical Computer Science. Birkhäuser, Boston (1991). <https://doi.org/10.1007/978-1-4684-6802-1>
- [Koi99] Koiran, P.: The real dimension problem is NPR-complete. *J. Complex.* **15**(2), 227–238 (1999)
- [Kut87] Kutylowski, M.: Small Grzegorzcyk classes. *J. Lond. Math. Soc.* **36**(2), 193–210 (1987)
- [KZ01] Kontsevich, M., Zagier, D.: Periods. In: Engquist, B., Schmid, W. (eds.) *Mathematics Unlimited – 2001 and Beyond*, pp. 771–808. Springer, Heidelberg (2001). https://doi.org/10.1007/978-3-319-50926-6_12
- [Mee06] Meer, K.: Optimization and approximation problems related to polynomial system solving. In: *Proceedings of the 2nd Conference on Computability in Europe (CiE 2006)*, pp. 360–367 (2006)
- [MU05] Mitzenmacher, M., Upfal, E.: *Probability and Computing - Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York (2005)
- [Mül01] Müller, N.T.: The iRRAM: exact arithmetic in C++. In: Blanck, J., Brattka, V., Hertling, P. (eds.) *CCA 2000*. LNCS, vol. 2064, pp. 222–252. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45335-0_14
- [MZ14] Müller, N.T., Ziegler, M.: From calculus to algorithms without errors. In: Hong, H., Yap, C. (eds.) *ICMS 2014*. LNCS, vol. 8592, pp. 718–724. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44199-2_107
- [Pap94] Papadimitriou, C.H.: *Computational Complexity*. Addison-Wesley, Boston (1994)
- [PP16] Popescu-Pampu, P.: What is the Genus?. *LMN*, vol. 2162. Springer, Cham (2016). <https://doi.org/10.1007/978-3-319-42312-8>
- [Rit63] Ritchie, R.W.: Classes of predictably computable functions. *Trans. Am. Math. Soc.* **106**(1), 139–173 (1963)
- [Sko08] Skordev, D.: On the subrecursive computability of several famous constants. *J. Univers. Comput. Sci.* **14**(6), 861–875 (2008)
- [SWG12] Skordev, D., Weiermann, A., Georgiev, I.: M^2 -computable real numbers. *J. Logic Comput.* **22**(4), 899–925 (2012)
- [Tur37] Turing, A.M.: On computable numbers, with an application to the “Entscheidungsproblem”. *Proc. Lond. Math. Soc.* **42**(2), 230–265 (1937)
- [TZ10] Tent, K., Ziegler, M.: Computable functions of reals. *Münster J. Math.* **3**, 43–65 (2010)
- [VS17] Viu-Sos, J.: A semi-canonical reduction for periods of Kontsevich-Zagier. [arXiv:1509.01097](https://arxiv.org/abs/1509.01097) (2017)
- [Wei00] Weihrauch, K.: *Computable Analysis*. Texts in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (2000). <https://doi.org/10.1007/978-3-642-56999-9>
- [Yos08] Yoshinaka, M.: Periods and elementary real numbers. [arXiv:0805.0349](https://arxiv.org/abs/0805.0349) (2008)

A Note on Online Colouring Problems in Overlap Graphs and Their Complements

Marc Demange¹ and Martin Olsen²(✉)

¹ School of Science, RMIT University, Melbourne, Australia
marc.demange@rmit.edu.au

² BTECH, Aarhus University, Aarhus, Denmark
martino@btech.au.dk

Abstract. We consider online versions of different colouring problems in interval overlap graphs, motivated by stacking problems. An instance is a system of time intervals presented in non-decreasing order of the left endpoints. We consider the usual colouring problem as well as b -bounded colouring and the same problems in the complement graph. We also consider the case where at most b intervals of the same colour can include the same element. For these versions, we obtain a logarithmic competitive ratio with respect to the maximum ratio of interval lengths. The best known ratio for the usual colouring was linear, and to our knowledge other variants have not been considered. Moreover, pre-processing allows us to deduce approximation results in the offline case. Our method is based on a partition of the overlap graph into permutation graphs, leading to a competitive-preserving reduction of the problem in overlap graphs to the same problem in permutation graphs. This new partition problem by itself is of interest for future work.

1 Introduction

The problem we consider is originally motivated by stacking problems (see, e.g., [1, 11]). After unloading a ship, containers are stored in the port before being uploaded again on another ship or truck. Due to space and logistics constraints, stored containers are stacked such that the container needing to leave first is preferably on the top of a stack, and in this work, we will consider the *perfect case* where this constraint is always satisfied. The problem is then to arrange the containers so as to minimise the space used on the floor. Additional constraints, like a maximum height of the stacks, are also natural. In a real context, the arrival time of ships and the final destination of their containers are not known or partially known in advance and one needs to stack the newly arrived containers without knowing details about the future ones. This constraint motivates the *online* version [14], where containers are supposed to arrive in any order and are allocated to stacks without taking into account future containers. A similar online model motivated by track assignment problems in railway optimisation

is studied in [3, 4] with a constraint on the number of trains per track. In this context, another natural restriction is the *midnight condition* that states that all trains are at the station at some time.

The stacking problem (in the perfect case) and the track assignment problem can be modelled as a colouring problem¹ in a specific class of graphs, called *overlap graphs*, representing incompatibilities between containers in a single stack (resp., trains on a track). Vertices correspond to the set R of time intervals during which containers are stored in the port and two vertices are linked if the related intervals *overlap*, i.e., they intersect but none is contained in the other. In this case, the related containers need to be in different stacks. R is referred to as an *interval system* and all intervals are assumed to have non negative left endpoints. If all intervals in a system intersect, the related graph is a *permutation graph* [6, 7], formally defined later. So, our work deals with *online colouring of overlap graphs and permutation graphs*. The cases where stacks have a maximum height and tracks have a maximum capacity motivate variants of graph colouring, precisely the *b-bounded load colouring* defined later and the *b-bounded colouring*, where colour classes cannot exceed the size b . In a theoretical perspective we address a generalisation, called *H-colouring*, for a hereditary property H ; it includes in particular *(b-bounded) clique covering* corresponding to *(b-bounded) colouring* in the complementary graph. Main definitions are given in the next section and for all graph notions not defined here, the reader is referred to [7].

Related work

For the offline version, 4-colourability has been shown NP-complete in overlap graphs [16]. Colouring permutation graphs can be done in polynomial time, even in an online set-up using a First-Fit strategy if the permutation is presented from left to right, as explained in the next section. However, the *b-bounded colouring* problem is NP-hard in permutation graphs for $b \geq 6$ [9]. Clique covering is also known to be hard in overlap graphs [10] and other NP-hard versions of stacking problems are considered in [1, 11].

To our knowledge, the best known polynomial approximation for colouring overlap graphs is a $(\log n)$ -approximation [2], where n is the number of vertices. It applies to the larger class of *subtree filament* graphs [10]. A 2-approximation in overlap graphs is claimed in [16], but it is contradicted in [11]. For clique covering overlap graphs, a $(\log \beta(G))$ -approximation is proposed in [10], where $\beta(G)$ denotes the optimal value for the graph G . This is improved in [15]: for an instance G_R defined by an interval system R , a $2(1 + \log \tilde{\alpha}(R))$ -approximation is proposed, where $\tilde{\alpha}(R)$ is the maximum number of pairwise disjoint intervals. If $\alpha(G_R)$ is the *independence number* of G_R , we have $\tilde{\alpha}(R) \leq \alpha(G_R) \leq \beta(G_R)$.

Online colouring of overlap graphs has mainly been considered in [4], where a $O(\frac{L}{\ell})$ -competitive algorithm is proposed if intervals are processed from *left to right*², L and ℓ being the largest and smallest interval lengths, respectively.

¹ Colour vertices using the minimum number of colours such that adjacent vertices have different colours.

² I.e., in non-decreasing order of their left endpoint.

It is improved into $O\left(\frac{\log^2 L}{\log \log L}\right)$ if ℓ is known in advance. Online colouring of permutation graphs, however, has been widely studied (see, e.g. [13]). In [4], online b -bounded colouring is considered in permutation graphs and a $(2 - 1/\sigma)$ -competitive algorithm is proposed, where σ is the minimum between b and the offline optimal value; moreover it is the best possible competitive ratio. For overlap graphs processed from left to right, it is shown [4] that a constant competitive ratio cannot be achieved. The proof can easily be transformed to show that a $K(\log \log n)$ -competitive ratio cannot be guaranteed for some constant K .

Our contribution

Here, we propose a $O(\log(\frac{L}{\ell}))$ -competitive online algorithm when intervals are processed from left to right; so an improvement from a linear to a logarithmic ratio and moreover, the result also holds for other H -colouring problems like clique covering, b -bounded colouring and b -bounded clique covering (Theorem 2) as well as for b -bounded load colouring (Theorem 3). These results are obtained using a partition of an overlap graph into permutation graphs, a new problem interesting by itself. It leads to a *competitive-preserving reduction* transforming an online algorithm in permutation graphs into an online algorithm in overlap graphs with a competitive ratio increased by a logarithmic factor (Theorem 1). This narrows the gap between the best competitive ratio and the best lower bound.

Even though we mainly focus on the online version, we conclude by showing how to use our online algorithms for the offline cases. We derive a $O(\log \tilde{\alpha}(R))$ -approximation for several colouring problems (Theorem 4). It slightly improves the best known approximation for colouring and clique covering in overlap graphs. Moreover, the same holds for the bounded versions, which constitutes, to our knowledge, the first approximations for these problems. It is worth noting that these approximation results are derived from online algorithms. Since a constant competitive ratio cannot be achieved, it highlights the question - left open - whether offline colouring is constant approximable in overlap graphs.

2 Preliminaries

\mathbb{N} , \mathbb{N}^* and \mathbb{Z} will denote the set of natural numbers, the set of positive natural numbers and the set of integers, respectively. In a graph G an *independent set* is a set of pairwise non-adjacent vertices while a *clique* is an independent set in the complement \overline{G} of G . The *independence number* and the *clique number* of G denote the maximum size of an independent set and of a clique in G , respectively.

Overlap and permutation graphs

All real intervals are assumed to have non-negative left endpoints and $]a, b[$ and $]c, d[$ *overlap* if $a < c < b < d$. Given a system R of open intervals, the related *overlap graph*, $G_R = (R, E_R)$, is defined such that adjacent vertices correspond to overlapping intervals [6]. The *interval graph* associated with R has the same vertex set but two vertices are linked if the related intervals intersect. We refer

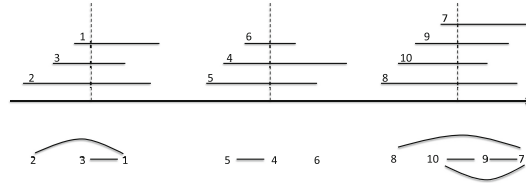


Fig. 1. BBQ arrangement B of intervals (above) and the related permutation graph (below) associated with the permutation $\pi_B = (2, 3, 1, 5, 4, 6, 8, 10, 9, 7)$ or equivalently, for instance, the list $Q_B = (0.5, 1.5, 0, 2.5, 2.1, 3.5, 4.5, 5.5, 5.3, 4.1)$.

to the clique number of this interval graph as the *load* of R , and we denote by $\tilde{\alpha}(R)$ its independence number.

Permutation graphs are usually defined from a permutation $\pi = (\pi_1, \dots, \pi_n)$. The vertex set of G_π is $\{\pi_1, \dots, \pi_n\}$ with an edge between π_i and π_j if $i < j$ and $\pi_i > \pi_j$ (see Fig. 1); we equivalently define G_Q from any list $Q = (q_1, \dots, q_n)$ of pairwise distinct numbers. It is a well known subclass of overlap graphs: in particular the overlap graph induced by a system of pairwise intersecting intervals (midnight condition) is a permutation graph (see, e.g. [4]). We call such a set of intersecting intervals a *brochette of intervals* and two brochettes are *independent* if every two intervals in different brochettes are disjoint; a set of pairwise independent brochettes is called a *BBQ arrangement*. Given a BBQ arrangement B with k brochettes B_1, \dots, B_k , we denote by $G_{B_i} = (V_i, E_i)$ the permutation graph associated with B_i . The overlap graph G_B associated with B is then the disjoint union of G_1, \dots, G_k , which is clearly a permutation graph:

Proposition 1. *BBQ arrangements always induce permutation graphs.*

Figure 1 gives an example of a BBQ arrangement B with the related permutation graph G_B , the corresponding permutation π and an example of a list Q_B satisfying $G_B = G_{Q_B}$ if we rename by q_i the vertex π_i , for $i = 1, \dots, n$. Q_B is chosen without positive integral values to avoid any confusion with values in π .

H-colouring problems and their online version

Graph colouring and its generalisations are extensively studied, in particular for their applications in scheduling. In the Minimum H -colouring problem, colour classes are constrained to satisfy a fixed hereditary property H . For overlap graphs, defined by an interval system R , suppose we are given a hereditary property H on interval systems, i.e., whenever a system satisfies H , so does any subsystem. Then, the Minimum H -colouring problem is to partition any interval system R into a minimum number of colour classes that all satisfy H . Table 1 lists the H -colouring problems we consider here and the related properties H .

Other examples of H -colouring include cocolouring [12] or split-colouring [5]. In the online version of colouring problems, vertices are presented one by one and each time a new vertex is presented, one needs to irrevocably decide its colour. If we number the vertices v_1, \dots, v_n in the order of presentation, an online colouring

Table 1. Examples of H -colouring problems in an overlap graph defined by R .

Problem	Property H satisfied by each colour class $R' \subset R$
<i>colouring</i>	$G_{R'}$ is an independent set (no overlap)
<i>b-bounded colouring</i>	$G_{R'}$ is an independent set and $ R' \leq b$
<i>clique covering</i>	$G_{R'}$ is a clique (every two intervals overlap)
<i>b-bounded clique covering</i>	$G_{R'}$ is a clique and $ R' \leq b$
<i>b-bounded load colouring</i>	$G_{R'}$ is an independent set and R' is of load at most b

algorithm decides the colour of vertex v_k using only the structure of the subgraph induced by vertices v_1, \dots, v_k and the colours of vertices v_1, \dots, v_{k-1} .

It is sometimes relevant to impose a specific order of presentation for the vertices. In this work, the instance is an overlap graph defined by a system of intervals and we assume that intervals are presented in non-decreasing order of their left endpoints, called the *left to right* order. For instance, in Fig. 1, the labelling of intervals, written from left to right, corresponds to the related permutation π_B . Note that π_B cannot be performed online but an equivalent list of numbers such as Q_B can.

Consequently, when considering a permutation graph defined by a BBQ arrangement presented from left to right, one can use any online algorithm using a list of numbers presented from left to right. In this case in particular, the greedy *First Fit algorithm*, that assigns the first possible colour to the new presented vertex, determines an optimal colouring (see, e.g., [4]).

The quality of an online algorithm is characterised by the *competitive ratio*. For a colouring problem, we denote by $\beta(G)$ the optimal value for G : it is the chromatic number $\chi(G)$ of G for the usual colouring problem, the bounded chromatic number $\chi_b(G)$ for the b -bounded colouring problem, or more generally $\chi_H(G)$ for the H -colouring problem. Given an online algorithm A for H -colouring, we denote by $\lambda(G, \mathcal{O})$ the value of the online solution computed for G presented in the order \mathcal{O} . A is said to guarantee the competitive ratio of ρ if we have $\lambda(G, \mathcal{O}) \leq \rho \times \beta(G)$ for any graph G and order \mathcal{O} within the considered model. Since we only consider the order from left to right, we will omit the parameter \mathcal{O} .

Even if it is not systematically considered in the literature, it is relevant to analyse the complexity of online algorithms. An online algorithm is said to be *polynomial* if each step can be performed in polynomial time with respect to the number of already presented vertices. In the offline case, the approximation ratio of a polynomial algorithm is defined in a similar way as the competitive ratio.

3 Partitioning an Overlap Graph into Permutation Graphs

Our general strategy is to partition an overlap graph into permutation sub-graphs: given an interval system R , we partition it into BBQ arrangements, called

clusters, \mathcal{P}_k , $k \in B \subset \mathbb{N}$, each inducing a permutation graph. As we will show, this can be done online when intervals are presented from left to right.

Note then that each cluster is presented from left to right, which will allow us to apply known online algorithms for permutation graphs on each part (See Sect. 4). Even though the complement of an overlap graph is not always an overlap graph, the fact that the class of permutation graphs is stable by complementation will allow us to derive similar results for clique covering.

To help understanding the main idea, we first describe a (offline) decomposition that requires the lengths L and ℓ . Then, in Proposition 2, we show that this decomposition can be performed online, and we also show how to handle the case where L is not known in advance (ℓ will never be requested).

The offline decomposition

Without loss of generality, we assume that the left endpoint of the first interval is 0. The decomposition works like a sieve based on nested discrete sets \mathcal{S}_i defined as follows. Let λ be the first interval length. We define $k_L = \lceil \log_2(\frac{L}{\lambda}) \rceil + 1$ and $k_\ell = \lceil \log_2(\frac{\ell}{\lambda}) \rceil + 1$. For $-k_L \leq i \leq k_\ell$, we set $\mathcal{S}_i = \{k2^{-i}\lambda, k \in \mathbb{N}^*\}$ and $\mathcal{S}_i = \emptyset$ for $i < -k_L$ or $i > k_\ell$. Note that $\mathcal{S}_{-k_L} \subset \dots \subset \mathcal{S}_0 \subset \dots \subset \mathcal{S}_{k_\ell}$.

We see sets \mathcal{S}_i as positions of vertical spikes of various lengths, as illustrated in Fig. 2. The spikes in \mathcal{S}_{-k_L} are the longest ones and \mathcal{S}_{i+1} is obtained by adding a shorter spike between every two spikes in \mathcal{S}_i . Using the BBQ metaphor, we see the full instance as a steak sliced into pieces of meat (intervals) to be dropped and skewered on the first (longest) spike corresponding to an element of \mathcal{S}_i for some i . The cluster \mathcal{P}_i is defined as the intervals skewed at level i : it is the set of intervals that intersect \mathcal{S}_i but do not intersect \mathcal{S}_{i-1} . Since $k_\ell > \log_2(\frac{\ell}{\lambda})$, we have for every $I \in R$, $I \cap \mathcal{S}_{k_\ell} \neq \emptyset$. So, no interval is left outside clusters.

We claim that each cluster \mathcal{P}_i can be partitioned into independent brochettes, thus is a BBQ arrangement, and consequently, the graph associated with \mathcal{P}_i is a permutation graph. Denoting $I =]a_I, b_I[$, we have indeed:

$$\forall I \in R : I \in \mathcal{P}_j \Leftrightarrow j = \min \left\{ i \in \{-k_L, \dots, k_\ell\} : \left\lfloor \frac{2^i a_I}{\lambda} \right\rfloor \leq \left\lfloor \frac{2^i b_I}{\lambda} \right\rfloor \right\} \quad (1)$$

If $j > -k_L$, Eq. (1) means that the interval $\left[\frac{2^j a_I}{\lambda}; \frac{2^j b_I}{\lambda} \right]$ includes an integer while interval $\left[\frac{2^{j-1} a_I}{\lambda}; \frac{2^{j-1} b_I}{\lambda} \right]$ does not; if $j = -k_L$, then $\left[\frac{2^{-k_L} a_I}{\lambda}; \frac{2^{-k_L} b_I}{\lambda} \right]$ includes an integer. Note that, for $i = -k_L$, the distance $2^{k_L} \lambda$ between two distinct elements of \mathcal{S}_{-k_L} is at least $2L$. Consequently,

$$\forall x_1, x_2 \in \mathcal{S}_i, I_1, I_2 \in \mathcal{P}_i : [x_1 \in I_1 \wedge x_2 \in I_2 \wedge x_1 \neq x_2] \Rightarrow I_1 \cap I_2 = \emptyset,$$

which proves the claim. A brochette in \mathcal{P}_i corresponds to an element $s \in \mathcal{S}_i \setminus \mathcal{S}_{i-1}$; in this case, (s, i) (called a skew) is said to be active.

Since $\mathcal{S}_i = \emptyset$ for $i < -k_L$ or $i > k_\ell$, there are at most $(k_L + k_\ell + 1) \leq (\lceil \log_2(\frac{L}{\lambda}) \rceil + 4)$ permutation graphs in the decomposition.

We know show that this decomposition can be performed online. Intervals are considered from left to right and Fig. 2 corresponds to the time $t = t_0$.

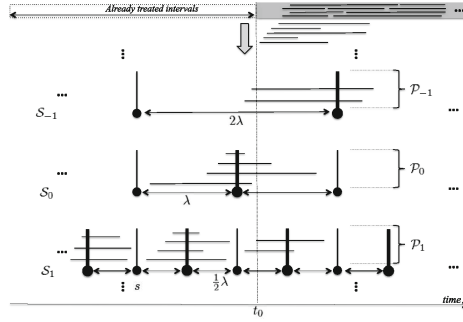


Fig. 2. The steak is sliced into intervals to be dropped on several layers \mathcal{P}_i of brochettes. Active skewers are thick, in particular $(s, 1), (s, 0)$ and $(s, -1)$ are inactive if $s \in \mathcal{S}_{-2}$. If the distance between the brochettes in the top layer is at least $2L$ then each \mathcal{P}_i is a BBQ arrangement.

Proposition 2 (*Partition into permutation graphs*). *Algorithm 1* described below is a polynomial online algorithm that partitions an overlap graph defined by an interval system presented from left to right into at most $(2 \lfloor \log_2(\frac{L}{\ell}) \rfloor + 7)$ permutation graphs defined by a BBQ arrangement.

Moreover, if L is known in advance, a simplified version of the algorithm guarantees a number of at most $(\lfloor \log_2(\frac{L}{\ell}) \rfloor + 4)$ permutation graphs in the decomposition.

Proof. As soon as a new interval is presented it will be assigned to a cluster corresponding to a BBQ arrangement. During this process a cluster will be called *open* if it is non-empty. If we decide to assign an interval to a not yet open cluster then we will open a new one for this interval. Note that, for a given L, ℓ, I , if the cluster \mathcal{P}_j of I is determined by Relation (1), then j 's value does not change if ℓ is replaced by a smaller value $\tilde{\ell}$. If L is replaced by a larger value \tilde{L} however, then j may change only if $j = -k_L$. Consequently, this decomposition can directly be performed online if L is known in advance with a maximum number of $(\lfloor \log_2(\frac{L}{\ell}) \rfloor + 4)$ BBQ-arrangements in the decomposition. In this case, the algorithm is simplified since the clusters \mathcal{R} are not required (the simplified version of the algorithm can be easily derived).

If L is not known in advance, we need to make a specific treatment for any interval I_0 placed in $\mathcal{P}_{-k_{L_0}}$ where L_0 is the largest known interval length when I_0 is revealed. Algorithm 1 shows how to perform the partition dynamically. To handle the above mentioned problem, it creates two kinds of clusters, \mathcal{Q} and \mathcal{T} . Roughly speaking a cluster $\mathcal{P}_{-k_{L_0}}$ in the previous decomposition might be doubled if L_0 is the largest known interval length when the cluster is open.

Line 6 ensures that, at any step of the online process, ℓ and L are respectively equal to the minimum and the maximum length of already revealed intervals. Once a set \mathcal{S}_i is defined as non-empty, its definition will not change. New non-empty sets \mathcal{S}_i are added when the values of L or ℓ are updated. Lines 10 and 12 determine in which cluster the interval should be put.

Consider first a cluster \mathcal{T}_{j_0} for some j_0 . Intervals in \mathcal{T}_{j_0} are assigned at lines 12, which means that $j_0 > -k_L$ each time an interval has been assigned to it. For any such interval $]a_I; b_I[$ the interval $\left[\frac{2^{j_0-1}a_I}{\lambda}; \frac{2^{j_0-1}b_I}{\lambda} \right]$ does not include any integer number, and consequently, neither do the intervals $\left[\frac{2^h a_I}{\lambda}; \frac{2^h b_I}{\lambda} \right]$ for $h < j_0$. Thus, Relation (1) will hold for any interval $I \in \mathcal{T}_{j_0}$ after the Algorithm stops and the previous analysis shows that \mathcal{T}_{j_0} is a BBQ arrangement of intervals.

Consider now a cluster \mathcal{Q}_{j_0} ; intervals have been assigned in this cluster at lines 10. Consider the last interval $I \in \mathcal{Q}_{j_0}$ revealed at the online step n_0 : denoting by L_0 the longest length of intervals revealed at steps $i \leq n_0$ we have $j_0 = -k_{L_0}$. All intervals in \mathcal{Q}_{j_0} are of length at most L_0 and intersect \mathcal{S}_{j_0} (see lines 8 and 10). The distance between any two elements in \mathcal{S}_{j_0} is at least $2^{-j_0}\lambda \geq 2L_0$. Since all intervals are open we deduce that \mathcal{Q}_{j_0} is a BBQ arrangement of intervals. All in all, the original overlap graph defined by an interval system is partitioned online into at most $(2k_L + k_\ell + 1) \leq (2 \lfloor \log_2(\frac{L}{\ell}) \rfloor + 7)$ BBQ arrangements of intervals, each revealed from left to right. Note finally that, at each step of the online process (new presented interval), the algorithm requires a linear complexity to decide in which cluster the interval should be added. It concludes the proof.

Algorithm 1. Partition into permutation graphs

Require: An overlap graph $G = (R, E_R)$ presented online from left to right (the maximum length L is not known in advance).

Ensure: A partition of R , $(\mathcal{P}_1, \dots, \mathcal{P}_p)$ such that $G[\mathcal{P}_i]$ is a permutation graph.

- 1: $\mathcal{T}_i \leftarrow \emptyset, i \in \mathbb{Z}$
 - 2: $\mathcal{Q}_i \leftarrow \emptyset, i \in \mathbb{Z}$
 - 3: $\ell \leftarrow \lambda, L \leftarrow \lambda$
 - 4: When the first interval I is presented, set λ as its length and add I to \mathcal{T}_0
 - 5: **for** each new interval $I =]a_I, b_I[$ **do**
 - 6: $\ell \leftarrow \min\{b_I - a_I, \ell\}, L \leftarrow \max\{b_I - a_I, L\}$
 - 7: $k_L \leftarrow \lceil \log_2(\frac{L}{\lambda}) \rceil + 1, k_\ell \leftarrow \lfloor \log_2(\frac{\lambda}{\ell}) \rfloor + 1$
 - 8: $j \leftarrow \min \left\{ i \in \{-k_L, \dots, k_\ell\} : \left\lceil \frac{2^i a_I}{\lambda} \right\rceil \leq \left\lfloor \frac{2^i b_I}{\lambda} \right\rfloor \right\}$
 - 9: **if** $j = -k_L$ **then**
 - 10: Add I to \mathcal{Q}_j
 - 11: **else**
 - 12: Add I to \mathcal{T}_j
 - 13: **end if**
 - 14: **end for**
 - 15: The final partition is $(\mathcal{T}_{-k_L+1}, \dots, \mathcal{T}_{k_\ell}) \cup (\mathcal{Q}_{-k_L}, \dots, \mathcal{Q}_0)$
-

4 Competitiveness Through Partitioning

Using Proposition 2, we reduce a large class of online H -colouring problems on overlap graphs to the same problem on permutation graphs:

Theorem 1 (*Online reduction*). *For any online algorithm for a H -Colouring problem guaranteeing a competitive ratio of ρ on permutation graphs defined by a BBQ arrangement presented from left to right, there is an online algorithm for the same problem on overlap graphs defined by an interval system presented from left to right guaranteeing the competitive ratio $(2 \lfloor \log_2(\frac{L}{\ell}) \rfloor + 7) \rho$. If L is known in advance the ratio is $(\lfloor \log_2(\frac{L}{\ell}) \rfloor + 4) \rho$.*

Moreover, if the former online algorithm is polynomial, then the latter is polynomial as well.

Proof. We can partition online the overlap graph into $(2 \lfloor \log_2(\frac{L}{\ell}) \rfloor + 7)$ permutation graphs defined by a BBQ arrangement using Proposition 2. If L is known in advance, then the number of parts is reduced to $(\lfloor \log_2(\frac{L}{\ell}) \rfloor + 4)$.

Note that each BBQ arrangement in the decomposition is presented from left to right. This allows us to apply the online algorithm for permutation graphs on each BBQ arrangement using different colour sets. This defines a feasible H -colouring of the whole graph. Since the H -chromatic number is not increasing from a graph to a subgraph, the result immediately follows. Note finally that, since Algorithm 1 requires at each step a linear complexity, this reduction transforms a polynomial online algorithm for permutation graphs into a polynomial online algorithm in overlap graphs, which concludes the proof.

Combining this reduction and known competitive algorithms in permutation graphs presented from left to right leads to the following result:

Theorem 2. *The following online competitive ratios can be guaranteed by a polynomial online algorithm in overlap graphs defined by an interval system presented from left to right:*

1. $2 \lfloor \log_2(\frac{L}{\ell}) \rfloor + 7$ for (unbounded) colouring and clique covering;
2. $(2 \lfloor \log_2(\frac{L}{\ell}) \rfloor + 7) \left(2 - \frac{1}{\min\{b, \beta(G_R)\}}\right)$ for b -bounded colouring and b -bounded clique covering, where $\beta(G_R)$ represents the offline optimal value in G_R .

Proof. Given a BBQ arrangement B presented from left to right, one can perform online a list Q_B defining the permutation graph G_{Q_B} (see Fig. 1). Moreover, the list Q_B^{-1} obtained by changing all signs in Q_B , also performed online, defines the complement graph $G_{Q_B^{-1}} = \overline{G_{Q_B}}$. Consequently, all results applying to online colouring permutation graphs also apply to online clique covering.

(1) For the usual colouring problem, the greedy algorithm First Fit is an online exact algorithm (1-competitive) for permutation graphs defined by a BBQ arrangement presented from left to right. Using the previous remark it gives as well an exact online algorithm for clique covering. We conclude using Theorem 1.

(2) For the online b -bounded colouring problem in a permutation graph G_Q associated with a list Q presented from left to right, an online algorithm guaranteeing the competitive ratio $2 - 1/\sigma$ is proposed in [4], where $\sigma = \min\{b, \chi_b(G_Q)\}$. We conclude the case of b -bounded colouring by noting that, if G_Q is a subgraph of an overlap graph G , we have: $2 - \frac{1}{\min\{b, \chi_b(G_Q)\}} \leq 2 - \frac{1}{\min\{b, \chi_b(G)\}}$. The same holds in the complementary graph $G_{Q_B^{-1}} = \overline{G_{Q_B}}$, which concludes the proof.

To our knowledge this is the first non-obvious competitive analysis for online b -bounded colouring of overlap graphs and their complements. Note finally that Theorem 1 can be applied for a problem defined directly on an interval system R like, for example, b -bounded load colouring.

Consider a BBQ arrangement B processed from left to right, the brochettes $B_i, i = 1, \dots, k$, can be determined online. An online solution for b -bounded load colouring can be performed using the algorithm in [4] on each brochette, and then, reusing the same colour set when passing to the next brochette. The overall number of colours is the maximum number of colours used in a single brochette and the same holds for a BBQ arrangement $B: \beta(B) = \max_{i=1, \dots, k} (\chi_b(G_{B_i}))$. Since $\min(b, \chi_b(G_{B_i})) \leq \min(b, \beta(B))$, for all $i = 1, \dots, k$, the algorithm in [4] gives a $\left(2 - \frac{1}{\min(b, \beta(B))}\right)$ -competitive algorithm for b -bounded load colouring in a BBQ arrangement. Using the same analysis as in Theorem 2-(2), we have:

Theorem 3. *There is a polynomial online algorithm for b -bounded load colouring in an interval system R presented from left to right with competitive ratio $\left(2 \lfloor \log_2(\frac{b}{\ell}) \rfloor + 7\right) \left(2 - \frac{1}{\min\{b, \beta(R)\}}\right)$, where $\beta(R)$ represents the related offline optimal value in R .*

5 Approximation of Offline Variants

We finally show how combining our online results with a preprocessing step gives new approximation results for various colouring problems in overlap graphs.

Proposition 3. *Given an interval system R of size n and $\varepsilon > 0$, we can modify R in $O(n \log n)$ -time, preserving the relative position of intervals (containment, overlapping and disjoint relation), so that in the new system R' , the maximum length $L(R')$ and the minimum length $\ell(R')$ satisfy $\frac{L(R')}{\ell(R')} \leq (2 + \varepsilon)\tilde{\alpha}(R)$.*

Proof. Given R , assume w.l.g. that the $2n$ endpoints are all distinct and sort them in increasing order as in [8]. It requires $O(n \log n)$ -time and since then the transformation is linear. The idea is to stretch and compress intervals without changing their relative position. Each time we move an endpoint of an interval, we may move accordingly other endpoints, keeping their relative order and a positive distance between two consecutive end-points.

Let $1 > \varepsilon > 0$ and define $\beta \leq \frac{\varepsilon}{\varepsilon+4}$. Now pick in polynomial time a set S with $\tilde{\alpha}(R)$ pairwise disjoint intervals; the complexity is $O(n)$ once the endpoints are sorted [8]. Stretch or compress intervals in S so that they all have length 1 and rearrange them such that the distance between the right endpoint of an interval and the left endpoint of the consecutive interval is β . We may move accordingly other endpoints of intervals in order to keep the same relative order as R .

Number intervals in S from left to right, $S = \{s_1, \dots, s_{|S|}\}$ with $|S| = \tilde{\alpha}(R)$ and denote by A the left endpoint of s_1 and by B the right endpoint of $s_{|S|}$. We now define three sub-intervals L_i, R_i and M_i for each $s_i \in S$, each of length β . L_i shares the left endpoint with s_i , R_i shares the right endpoint with s_i and M_i is on

the middle of s_i . In other words, L_i and R_i are small zones in the left and right part of s_i , respectively, while M_i is a small zone in the middle. Since $\beta < \frac{1}{3}$, these three zones are disjoint. We first consider intervals with the right endpoint greater than B or the left endpoint less than A (note there is no left endpoint on the right of B neither right endpoint on the left of A by optimality of S). We compress these intervals so that all endpoints in the system are between $A - \frac{\beta}{2}$ and $B + \frac{\beta}{2}$; this can be done in linear time. Now we consider all intervals that are contained in one of the intervals s_i in S . We refer to this set of intervals as H_i . Two intervals in H_i intersect by optimality of S ; so their largest left endpoint is smaller than their smallest right endpoint. For every i we now move all the left endpoints of intervals in H_i to L_i without changing the relative order of endpoints in the whole system. We move the right endpoints to R_i in the same manner. Finally all the endpoints that are in some $s_i \setminus (L_i \cup R_i)$ (these endpoints have not been moved until now) are moved into M_i without changing their relative position. All in all, computing these modifications requires linear time.

We claim that the new system R' satisfies $\frac{L(R')}{\ell(R')} \leq (2 + \varepsilon)\tilde{\alpha}(R)$. Indeed, note first that $L(R') \leq |B - A| + \beta = \tilde{\alpha}(R)(1 + \beta)$. Consider now an interval I in R' of length less than $(1 - 2\beta)$ (if any). I cannot be contained in an interval s_i due to modifications performed in H_i and it cannot have its two endpoints in $[A - \frac{\beta}{2}, B + \frac{\beta}{2}] \setminus \cup_i s_i$ (outside the zone covered by S) by optimality of S . So, it necessarily overlaps one interval $s_i \in S$ and by optimality of S it overlaps all intervals in H_i as well. We deduce that I has one of its endpoints in M_i and the other one outside s_i . Thus, I is of length at least $\frac{1}{2} - \frac{\beta}{2}$, which means that $\ell(R') \geq \frac{1}{2} - \frac{\beta}{2}$. Consequently, $\frac{L(R')}{\ell(R')} \leq 2\tilde{\alpha}(R) \left(\frac{1+\beta}{1-\beta}\right) \leq (2+\varepsilon)\tilde{\alpha}(R)$ since $\frac{4\beta}{1-\beta} \leq \varepsilon$. This concludes the proof.

We deduce approximation results using our online results with L a priori known. For clique covering overlap graphs, it improves by a factor 2 the ratio in [15].

Theorem 4. *For an interval system R , minimum colouring and clique covering G_R are $(\log \tilde{\alpha}(R) + c)$ -approximable, for a constant c . This ratio is doubled for b -bounded colouring and b -bounded clique covering G_R as well as b -bounded load colouring of R .*

6 Concluding Remarks

Our contribution is twofold. First, we highlight the problem of partitioning an overlap graph into permutation graphs that seems interesting by itself. It is similar to the partition into layers in [2] but to our knowledge, it has not been considered in an online set-up until now. Our results motivate this approach that leads to improved competitive algorithms. Surprisingly, it leads to improved approximation algorithms in the offline case as well. The main advantage is to reduce colouring problems in overlap graphs to the same problems in permutation graphs. Since the partition can be done online, it leads to competitive-preserving



reductions. Since the class of permutation graphs is stable by complement while the class of overlap graphs is not, this approach allows to solve the same problem in the complement of an overlap graph as well. This method could be used also for other combinatorial problems. Our results highlight two questions we leave open for the offline case: can we partition an overlap graph in less than a logarithmic factor of permutation graphs and are the considered problems constant approximable in overlap graphs?

Acknowledgments. The authors would like to thank anonymous referees for their helpful comments.

References

1. Avriel, M., Penn, M., Shpirer, N.: Container ship stowage problem: complexity and connection to the coloring of circle graphs. *Discret. Appl. Math.* **103**(1–3), 271–279 (2000)
2. Černý, J.: Coloring circle graphs. *Electron. Notes Discret. Math.* **29**, 457–461 (2007)
3. Cornelsen, S., Di Stefano, G.: Track assignment. *J. Discret. Algorithms* **5**(2), 250–261 (2007)
4. Demange, M., Di Stefano, G., Leroy-Beaulieu, B.: On the online track assignment problem. *Discret. Appl. Math.* **160**(7–8), 1072–1093 (2012)
5. Ekim, T., de Werra, D.: On split-coloring problems. *J. Combin. Optim.* **10**, 211–225 (2005)
6. Gavril, F.: Algorithms for a maximum clique and a maximum independent set of a circle graph. *Networks* **3**(3), 261–273 (1973)
7. Golombic, M.C.: *Algorithmic Graph Theory and Perfect Graphs* (Annals of Discrete Mathematics, vol. 57). North-Holland Publishing Co., Amsterdam (2004)
8. Gupta, U.I., Lee, D.T., Leung, J.Y.-T.: Efficient algorithms for interval graphs and circular-arc graphs. *Networks* **12**(4), 459–467 (1982)
9. Jansen, K.: The mutual exclusion scheduling problem for permutation and comparability graphs. *Inf. Comput.* **180**, 71–81 (2003)
10. Keil, J.M., Stewart, L.: Approximating the minimum clique cover and other hard problems in subtree filament graphs. *Discret. Appl. Math.* **154**(14), 1983–1995 (2006)
11. König, F.G., Lübbecke, M.E.: Sorting with complete networks of stacks. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) *ISAAC 2008*. LNCS, vol. 5369, pp. 895–906. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-92182-0_78
12. Lesniak, L., Straight, H.J.: The chromatic number of a graph. *Ars Combin.* **3**, 39–46 (1977)
13. Nikolopoulos, S.D., Papadopoulos, C.: On the performance of the first-fit coloring algorithm on permutation graphs. *Inf. Process. Lett.* **75**(6), 265–273 (2000)
14. Olsen, M., Gross, A.: Probabilistic analysis of online stacking algorithms. In: Corman, F., Voß, S., Negenborn, R.R. (eds.) *ICCL 2015*. LNCS, vol. 9335, pp. 358–369. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24264-4_25
15. Shahrokhi, F.: A new upper bound for the clique cover number with applications. arXiv preprint [arXiv:1502.06168](https://arxiv.org/abs/1502.06168) (2015)
16. Unger, W.: On the k-colouring of circle-graphs. In: Cori, R., Wirsing, M. (eds.) *STACS 1988*. LNCS, vol. 294, pp. 61–72. Springer, Heidelberg (1988). <https://doi.org/10.1007/BFb0035832>

Online Facility Assignment

Abu Reyan Ahmed¹  , Md. Saidur Rahman², and Stephen Kobourov¹

¹ Department of Computer Science, University of Arizona, Tucson, USA
abureyanahmed@email.arizona.edu, kobourov@cs.arizona.edu

² Department of Computer Science and Engineering,
Bangladesh University of Engineering and Technology, Dhaka, Bangladesh
saidurrahman@cse.buet.ac.bd

Abstract. We consider the online facility assignment problem, with a set of facilities F of equal capacity l in metric space and customers arriving one by one in an online manner. We must assign customer c_i to facility f_j before the next customer c_{i+1} arrives. The cost of this assignment is the distance between c_i and f_j . The total number of customers is at most $|F|l$ and each customer must be assigned to a facility. The objective is to minimize the sum of all assignment costs. We first consider the case where facilities are placed on a line so that the distance between adjacent facilities is the same and customers appear anywhere on the line. We describe a greedy algorithm with competitive ratio $4|F|$ and another one with competitive ratio $|F|$. Finally, we consider a variant in which the facilities are placed on the vertices of a graph and two algorithms in that setting.

1 Introduction

Let $F = \{f_1, f_2, \dots, f_{|F|}\}$ be a set of facilities, each with capacity l . We first consider the case when facilities are placed on line L , such that the distance between every pair of adjacent facilities is d , where d is a constant. An input sequence $I = \{c_1, c_2, \dots, c_n\}$ is a set of n customers who arrive one at a time in an online manner, with c_i corresponding to the location of customer i on the line L . The distance between a customer c_i and a facility f_j is the Euclidean distance between c_i and f_j . We later consider the case in which the facilities are located on the vertices of a graph $G = (V, E)$ and customers appear on the vertices of G . In that case, the distance between a customer c_i and a facility f_j is equal to the number of edges in the shortest path between c_i and f_j .

Any algorithm for this problem must assign a customer c_i to a facility f_j before the next customer c_{i+1} arrives, where the cost of that assignment is the distance between c_i and f_j . The total number of customers is at most $|F|l$ as every facility can serve at most l customers and each customer must be assigned to a facility. The objective is to minimize the total cost of all assignments. We call this problem the *online facility assignment problem*. This problem arises

Work on this project was funded in part by NSF grant CCF-AF 1712119.

naturally in different practical applications, such handling online orders for a restaurant with multiple locations, and handling network packets in network with multiple routers.

1.1 Related Work

In the classical *facility location problem*, customer locations are known ahead of time and the objective is to compute locations for a set of facilities that can handle all the customers. The Fermat-Weber problem is considered the first facility location problem, studied as early as in the 17th century; see the survey of Drezner [9] and the textbook by Drezner and Hamacher [10]. A recently proposed facility location variant is the r -gathering problem. An r -gathering of a set of customers C for a set of facilities F is an assignment of C to open facilities $F' \subset F$ such that r or more customers are assigned to each open facility. Armon [3] describes a simple 3-approximation algorithm for this problem. Akagi and Nakano [1] provide an $O((|C| + |F|) \log |C| + |F|)$ time algorithm to solve the r -gathering problem when all customers in C and facilities in F are on the real line.

The online facility assignment problem is also related to the k -server problem proposed by Manasse et al. [16], which requires scheduling the movement of a set of k servers, represented as points in a metric space, in order to handle requests that are also in the form of points in the space. For each request, the algorithm must determine which server to move to the requested point, with the goal of minimizing the total distance covered by all servers. This problem has been extensively studied [5–8, 14, 15, 18].

Despite similarities, the k -server problem and the online facility assignment problem are different. The servers in the k -server problem are movable, whereas the positions of facilities are fixed in the online facility assignment problem. Therefore, a customer placed very close to a previous customer is easily served in the server problem which is not true for the facility assignment problem.

The facility assignment problem is also related to the matching problem [17], which is one of the fundamental and well-studied optimization problems. The facility assignment problem can be seen as a generalization of the matching problem, where each facility has capacity $l \geq 1$. Online variants of matching have been extensively studied [2, 4, 11–13]. Kao et al. [12] provide a randomized lower bound of 4.5911 for online matching on a line. We provide a randomized algorithm, which is $\frac{9}{2}$ -competitive for a class of input sequences. Antoniadis et al. [2] describe an $o(n)$ -competitive deterministic algorithm for online matching on a line.

1.2 Our Contributions

We first consider the case where both the facilities F and the customers C are on a line. We propose Algorithm Greedy and show that it has competitive ratio $4|F|$. Introducing randomization in Algorithm Greedy leads to an improved

performance of $9/2$ for a special class of input instances. We then describe Algorithm Optimal-Fill and show it has competitive ratio $|F|$.

We next consider the case where both the facilities F and the customers C are located on the vertices of an unweighted graph $G = (V, E)$. We show that Algorithm Greedy has competitive ratio $2|E(G)|$ and Algorithm Optimal-Fill has competitive ratio $|E(G)||F|/r$, where r is the radius of G . Finally, we consider the case where a customer leaves after receiving service at a facility. We define *service time* as the amount of service time needed, and study the facility assignment problem with limited service time.

The rest of this paper is organized as follows. In Sect. 2 we provide basic definitions. In Sect. 3 we study the online facility assignment problem on a line. In Sect. 4 we study the graph version of the problem. In Sect. 5 we introduce a service time parameter t in our model and show that no deterministic algorithm is competitive when $t = 2$.

2 Preliminaries

A graph $G = (V, E)$ consists of a finite set V of vertices and a finite set E of edges; each edge is an unordered pair of vertices. We often denote the set of vertices G by $V(G)$ and the set of edges by $E(G)$. We say G is *unweighted* if every edge of G has equal weight. Let u and v be two vertices of G . If G has a u, v -path, then the distance from u to v is the length of a shortest u, v -path, denoted by $d_G(u, v)$ or simply by $d(u, v)$. If G has no u, v -path then $d(u, v) = \infty$. The *eccentricity* of a vertex u in G is $\max_{v \in V(G)} d(u, v)$ and denoted by $\epsilon(u)$. The *radius* r of G is $\min_{u \in V(G)} \epsilon(u)$ and the *diameter* of G is $\max_{u \in V(G)} \epsilon(u)$. The *center* of G is the subgraph of G induced by vertices of minimum eccentricity.

In the online facility assignment problem, we are given a set of facilities $F = \{f_1, f_2, \dots, f_{|F|}\}$ of equal capacity l in a metric space, and an input sequence of customers $I = \{c_1, c_2, \dots, c_n\}$ which is a set of n customers who arrive one at a time in an online manner, with c_i corresponding to the location of customer i in the given space. We say an input I is *well distributed* if there is at least one customer between any two adjacent facilities. The capacity of a facility is reduced by one when a customer is assigned to it. We denote the current capacity of facility f_i by *capacity_i*. A facility f_i is called *free* if *capacity_i* > 0 . Any algorithm ALG for this problem must assign a customer c_i to a free facility f_j before a new customer c_{i+1} arrives. The cost of this assignment is the distance between c_i and f_j , which is denoted by *distance*(f_j, c_i). The total number of customers is, at most, $|F|l$ and each customer must be assigned to a facility. For any input sequence of customers I , $\text{Cost_ALG}(I)$ is defined as the total cost of all assignments made by ALG. The objective is to minimize $\text{Cost_ALG}(I)$.

We say an algorithm is *optimal* if, for any input sequence of customers, the total cost of the assignment it provides is the minimum possible. We denote an optimal algorithm by OPT. An online algorithm ALG is c -competitive if there is a constant α such that, for all finite input sequences I ,

$$\text{Cost_ALG}(I) \leq c \cdot \text{Cost_OPT}(I) + \alpha.$$

The factor c is called the *competitive ratio* of ALG. When the *additive constant* α is less than or equal to zero (i.e., $\text{Cost_ALG}(I) \leq c \cdot \text{Cost_OPT}(I)$), we may say, for emphasis, that ALG is *strictly* c -competitive. An algorithm is called *competitive* if it attains a constant competitive ratio c . Although c may be a function of the problem parameters, it must be independent of the input I . The infimum over the set of all values c such that ALG is c -competitive is called *the competitive ratio* of ALG and is denoted by $\mathcal{R}(\text{ALG})$.

3 Facility Assignment on a Line

Let $F = \{f_1, f_2, \dots, f_{|F|}\}$ be a set of facilities placed on a line, such that the distance between every pair of adjacent facilities is d , where d is a constant. An input sequence $I = \{c_1, c_2, \dots, c_n\}$ is a set of n customers who arrive one at a time in an online manner, with c_i corresponding to the location of customer i on the line. In Sect. 3.1 we describe Algorithm Greedy with competitive ratio $4|F|$. In Sect. 3.2 we introduce randomization to Algorithm Greedy and show that it is $\frac{9}{2}$ -competitive for a special class of input sequences. In Sect. 3.3 we describe Algorithm Optimal-Fill and show it has competitive ratio $|F|$.

3.1 Algorithm Greedy

Here we describe and analyze the natural greedy algorithm, which assigns each customer to the nearest free facility.

Algorithm Greedy

Input: Customers $I = \{c_1, \dots, c_n\}$, facilities $F = \{f_1, \dots, f_{|F|}\}$, capacity l

Output: An assignment of C to F and the total cost of that assignment

$sum \leftarrow 0$;

for $i \leftarrow 1$ **to** $|F|$ **do**

$capacity_i \leftarrow l$;

for $i \leftarrow 1$ **to** n **do**

$min \leftarrow \infty$;

$index \leftarrow -1$;

for $j \leftarrow 1$ **to** f **do**

if $capacity_j > 0$ **and** $distance(f_j, c_i) < min$ **then**

$min \leftarrow distance(f_j, c_i)$;

$index \leftarrow j$;

 assign c_i to f_{index} ;

$capacity_{index} \leftarrow capacity_{index} - 1$;

$sum \leftarrow sum + min$;

Result: sum is the total cost

We can analyze the online algorithm in the context of a game between an *online player* and a malicious *adversary*. The online player runs the online

algorithm on an input created by the adversary. The adversary, based on the knowledge of the online algorithm, constructs the worst possible input (i.e., one that maximizes the competitive ratio). Consider Algorithm Greedy above and the adversary strategy of making an instance very costly for Algorithm Greedy but, at the same time, inexpensive for OPT. The following lemma gives a lower bound for OPT's cost.

Lemma 1. *Let d be the distance between all adjacent facilities. If the assignments of OPT and Algorithm Greedy are not the same, then OPT's cost is at least $\frac{d}{2}$.*

Proof. Let c_x be the first customer for which the assignments of OPT and Algorithm Greedy differ. The optimal cost for assigning c_x is at least $\frac{d}{2}$. Hence the total optimal cost is at least $\frac{d}{2}$. \square

The following theorem determines the worst input sequence an adversary can construct for Algorithm Greedy and provides a competitive ratio.

Theorem 1. *Let $F = \{f_1, f_2, \dots, f_{|F|}\}$ be a set of facilities placed on the line, such that the distance between every pair of adjacent facilities is d , where d is a constant. Then $\mathcal{R}(\text{Algorithm Greedy}) \leq 4|F|$.*

Proof. Recall the definition of a well distributed input sequence, namely that there is at least one customer between any two adjacent facilities. When the metric space is a line, all customers have cost less than d in the optimum assignment of a well distributed input sequence. However, if the input sequence is not well distributed, there are some customers with assignment cost greater than d . We consider these two cases separately. For both cases, assume now that the facilities have unit capacity. Later we will also deal with the case for capacity l , where $l > 1$.

Let f_l be the leftmost facility and f_r be the rightmost facility. Consider a customer c who appears to the left of f_l . The distance between c and f_l is $\text{distance}(f_l, c)$. Both $\text{Cost_Algorithm_Greedy}(I)$ and $\text{Cost_OPT}(I)$ must pay the amount $\text{distance}(f_l, c)$. The ratio of $\text{Cost_Algorithm_Greedy}(I)$ to $\text{Cost_OPT}(I)$ increases when $\text{distance}(f_l, c)$ decreases. The case when c appears to the right of f_r is analogous. Now consider the case where customers appear between f_l and f_r , since the ratio does not increase if customers appear outside of this range (because both OPT and Algorithm Greedy have to consider the region outside this range).

We first consider the case when all customers have costs less than d in the optimum assignment. In the worst case, the adversary places all the customers very close to the facilities except the first customer c_1 as illustrated in Fig. 1. The total cost of Algorithm Greedy is no more than $2|F|d$. In the optimum assignment all customers c_i have cost ε_i except c_1 . The cost of the first customer c_1 is γ , where $\gamma > \frac{d}{2}$ (Lemma 1). Then $\frac{\text{Cost_Algorithm_Greedy}(I)}{\text{Cost_OPT}(I)} \leq \frac{2|F|d}{\frac{d}{2}} = 4|F|$.

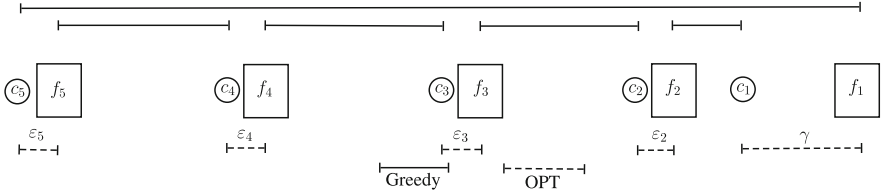


Fig. 1. The configurations of Algorithm Greedy and OPT

In the second case, k customers have costs greater than d in the optimum assignment. Hence, the total cost of the optimum assignment is at least kd . We have assumed that the customers at distance less than d are assigned with cost zero by the optimal algorithm and there are $|F| - k$ such customers. In the assignment created by Algorithm Greedy, each of these customers would have cost at most d . Note that if any of these customers have cost greater than d , then that assignment A can be easily transformed to an equivalent assignment B with total cost equal to that of the original assignment A and so that $|F| - k$ customers have cost no more than d . The transformation goes one step at a time, as follows. If a customer c_1 assigned to a facility f_1 by OPT has cost less than or equal to d and c_1 is assigned to a facility f_2 in A and has cost greater than d , then we get a new assignment A' by assigning c_1 to f_1 and c_2 to f_2 , where c_2 was the customer assigned to f_1 in A . Similarly, we can swap the next pair to get assignment A'' , and continue this process until we get the equivalent assignment B . There are $|F| - k$ customers in B with cost at most d and each of the remaining k customers have a cost at most $(|F| - 1)d$. Then
$$\frac{\text{Cost_Algorithm_Greedy}(I)}{\text{Cost_OPT}(I)} \leq \frac{(|F|-1)dk + (|F|-k)d}{kd} = \frac{(k+1)|F|}{k} - 2.$$

In the analysis above we assumed unit capacity; now let each facility have capacity l , where $l > 1$. Suppose that there exists an input sequence of customers I for which the ratio is greater than $4|F|$. We can partition I into I_1, I_2, \dots, I_l in such a way that the following conditions hold:

- $I_i \cap I_j = \emptyset$ for $1 \leq i, j \leq l$ and $i \neq j$.
- $I_1 \cup I_2 \cup \dots \cup I_l = I$.
- Exactly one customer from I_i is assigned to a facility f_j for $1 \leq i \leq l$ and $1 \leq j \leq |F|$.

Then there exists a set $I_{max} \in \{I_1, I_2, \dots, I_l\}$ such that the ratio of the corresponding cost of Algorithm Greedy to the cost of OPT is greater than $4|F|$. If we take a set of facilities with unit capacities and place the customers of I_{max} in the same order as they appear in I , the ratio would be greater than $4|F|$ which is a contradiction to the bound of unit capacity. \square

Note that this algorithm does not generalize to non-equidistant facilities. In particular, if the distances between adjacent facilities increase exponentially, this algorithm can be forced to pay a factor of $O(2^{|F|})$ more than OPT.

3.2 Algorithm σ -Randomized-Greedy

In this section we introduce randomness to the greedy method of the previous section and show that better competitive ratios can be obtained. With deterministic online algorithms, the adversary knows the full strategy and can select the worst input sequence. This is not possible if ALG is a randomized algorithm. An oblivious adversary must choose a finite input sequence I in advance. ALG is c -competitive against an oblivious adversary if for every such I , $E[\text{Cost_ALG}(I)] \leq c \cdot \text{Cost_OPT}(I) + \alpha$ where α is a constant independent of I , and $E[\cdot]$ is the mathematical expectation operator taken with respect to the random choices made by ALG. Since the offline player does not know the outcomes of the random choices made by the online player, $\text{Cost_OPT}(I)$ is not a random variable and there is no need to take its expectation.

We introduce randomness in Algorithm Greedy, described in the previous section, and call the new method Algorithm σ -Randomized-Greedy. Let f_x be the facility which is nearest to customer c_y and let σ be a real number. Then σ -Randomized-Greedy checks whether the distance between c_y and f_x is less than σ and if so then c_y is assigned to f_x . Otherwise, σ -Randomized-Greedy tosses a fair coin before assigning a customer to a facility, choosing the nearest free facility to the right (left) if the coin comes heads (tails).

We will next show that Algorithm σ -Randomized-Greedy performs better than Algorithm Greedy.

Algorithm σ -Randomized-Greedy

Input: Customers $I = \{c_1, \dots, c_n\}$, facilities $F = \{f_1, \dots, f_{|F|}\}$, capacity l

Output: An assignment of C to F and the total cost of that assignment
 $sum \leftarrow 0$;

for $i \leftarrow 1$ **to** $|F|$ **do**

$capacity_i = l$;

for $i \leftarrow 1$ **to** n **do**

$min \leftarrow \infty$;

$index \leftarrow -1$;

for $j \leftarrow 1$ **to** $|F|$ **do**

if $capacity_j > 0$ **and** $distance(f_j, c_i) < min$ **then**

$min \leftarrow distance(f_j, c_i)$;

$index \leftarrow j$;

if $min \geq \sigma$ **then**

 randomly select the nearest free facility f_k to the left or right;

$min \leftarrow distance(f_k, c_i)$;

$index \leftarrow k$;

 assign c_i to f_{index} ;

$capacity_{index} \leftarrow capacity_{index} - 1$;

$sum \leftarrow sum + min$;

Result: sum is the total cost

Theorem 2. *Let I be a well distributed request sequence for Algorithm Greedy. Let γ be the optimal cost for the first customer and ε_i be the optimal cost for i^{th} customer where $i > 1$. If $\sigma > \varepsilon_i$ for all i and $\sigma \leq \gamma$ then Algorithm σ -Randomized-Greedy is $\frac{9}{2}$ -competitive for I .*

Proof. Let $F = \{f_1, f_2, \dots, f_{|F|}\}$ be a set of facilities, such that the distance between every pair of adjacent facilities is d , where d is a constant. Recall that if an input I of customers has the property that all assignments cost less than d in the optimum solution, then I is *well distributed*. The first customer c_1 is placed closer to f_2 (Fig. 1) in order to fool Algorithm Greedy. Algorithm Greedy assigns c_1 to f_2 . Except the first customer c_1 , the adversary places every customer c_k very close to facility f_k . Since Algorithm Greedy has already assigned c_1 to f_2 , it can not assign c_2 to the same facility. Similarly, for every customer c_k , Algorithm Greedy assigns it to f_{k+1} although it is very close to f_k . Algorithm σ -Randomized-Greedy overcomes this situation by using randomness. Consider the first customer c_1 who is close to f_2 . Algorithm σ -Randomized-Greedy chooses either f_1 or f_2 with equal probability $\frac{1}{2}$. Similarly for every customer c_k , Algorithm σ -Randomized-Greedy chooses either f_{k+1} or f_k with equal probability $\frac{1}{2}$. Then

$$\begin{aligned} E[\text{Cost}_{\sigma\text{-Randomized-Greedy}}(I)] &= \frac{d}{4} + \sum_{i=1}^{|F|-2} \left\{ \frac{1}{2^{i+1}} (2id - \frac{d}{2}) \right\} \\ &\quad + \frac{1}{2^{|F|-1}} \left\{ 2(|F| - 1)d - \frac{d}{2} \right\} \\ &< \frac{d}{4} + d \sum_{i=1}^{|F|-2} \frac{i}{2^i} \\ &< \frac{d}{4} + 2d \\ &= \frac{9d}{4} \end{aligned}$$

Since the optimum cost is at least $d/2$, Algorithm σ -Randomized-Greedy is $\frac{9}{2}$ -competitive for I . □

This shows that Algorithm σ -Randomized-Greedy can obtain better (expected) competitive ratios than Algorithm Greedy, for appropriate values of σ . In the theorem above the value of σ is very small compared to d . If a customer c_i is placed beside a facility f_j such that the distance between c_i and f_j is less than σ , then it is assumed that there is no harm to assign c_i to f_j .

3.3 Competitive Analysis of Algorithm Optimal-Fill

In Sect. 3.1, we showed that Algorithm Greedy can be easily fooled by placing all the customers very close to the facilities except for the first customer. We next describe Algorithm Optimal-Fill, which is more efficient than Algorithm Greedy. The idea is that when a new customer c_i arrives, Algorithm Optimal-Fill finds out

facility f_j that would be selected by an optimal assignment of all the customers c_1, c_2, \dots, c_i . Algorithm Optimal-Fill then assigns c_i to f_j .

Algorithm Optimal-Fill

Input: Customers $I = \{c_1, \dots, c_n\}$, facilities $F = \{f_1, \dots, f_{|F|}\}$, capacity l

Output: An assignment of C to F and the total cost of that assignment
 $sum \leftarrow 0$;

for $i \leftarrow 1$ **to** n **do**
 let f_j be the new facility chosen by an optimal assignment of
 customers c_1, c_2, \dots, c_i ;
 assign c_i to f_j ;
 $sum \leftarrow sum + distance(f_j, c_i)$;

Result: sum is the total cost

The following theorem shows that Algorithm Optimal-Fill performs better than deterministic greedy method.

Theorem 3. *Let $F = \{f_1, f_2, \dots, f_{|F|}\}$ be a set of facilities placed on the line, such that the distance between every pair of adjacent facilities is d , where d is a constant. Then $\mathcal{R}(\text{Algorithm Optimal-Fill}) \leq |F|$.*

Proof. In the worst case, the adversary can place each customer except the first one on top of a facility, so the cost is zero, while Algorithm Optimal-Fill has to pay for each of these customers. The adversary pays only for the first customer and all others are free, because they are placed on top of their facilities. However, Algorithm Optimal-Fill has to pay at least d for each of them. The two algorithms (OPT and Optimal-Fill) are illustrated with an example with 5 facilities and 5 customers in Fig. 2.

Then $\frac{\text{Cost_Algorithm_Optimal-Fill}(I)}{\text{Cost_OPT}(I)} = \frac{d+2d+\dots+(|F|-1)d+\frac{d}{2}}{\frac{|F|d}{2}} < |F|$ □

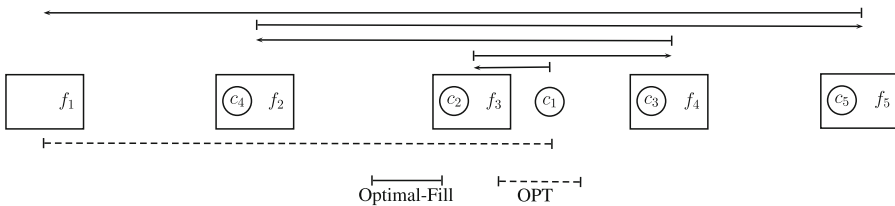


Fig. 2. The adversary places the first customer c_1 between f_3 and f_4 . Algorithm Optimal-Fill assigns c_1 to f_3 because it is a little bit closer to f_3 compared to f_4 . The adversary now places c_2 exactly on f_3 . Algorithm Optimal-Fill assigns c_2 to f_4 because f_3 and f_4 are chosen by an optimal assignment for customers c_1 and c_2 . The adversary then places c_3 exactly on f_4 . Algorithm Optimal-Fill assigns c_3 to f_2 because f_2 is the new facility chosen by an optimal assignment for customers c_1, c_2 and c_3 .

Note that $\mathcal{R}(\text{Algorithm Optimal-Fill})$ is not affected when the distances between adjacent facilities are different.

4 Facility Assignment on Connected Unweighted Graphs

We now consider the case where the facilities F are placed on the vertices of a connected unweighted graph $G = (V, E)$ and customers arrive one by one in an online manner at vertices of G . We show that Algorithm Greedy has competitive ratio $2|E(G)|$ and Algorithm Optimal-Fill has competitive ratio $|E(G)||F|/r$, where r is the radius of G .

4.1 Competitive Analysis of Algorithm Greedy

In Sect. 3.1 we analyzed Algorithm Greedy on a line. The following theorem describes the performance of Algorithm Greedy in the graph setting.

Theorem 4. *Let \mathcal{M} be a connected unweighted graph. Then $\mathcal{R}(\text{Algorithm Greedy}) \leq 2|E(\mathcal{M})|$.*

Proof. We assume that the facilities have unit capacity since the analysis is similar for capacity l , where $l > 1$. Two facilities f_i and f_j are *adjacent* if there exists a path P from f_i to f_j such that no other facilities are situated on P . Recall the definition of a well distributed input sequence: an input I is *well distributed* if there is at least one customer between any two adjacent facilities. We first prove the claim for an input I which is well distributed. Then we show how to transform I to I' such that I' is well distributed and the competitive ratios of I and I' are the same.

We consider two cases; \mathcal{M} is a tree and \mathcal{M} contains at least one cycle. If \mathcal{M} is a tree, we assume that every leaf contains a facility, since $\mathcal{R}(\text{Algorithm Greedy})$ does not increase in the other case. In the worst case $\text{Cost_Greedy}(I)$ is less than $2|E(\mathcal{M})|$ and $\text{Cost_OPT}(I)$ is equal to one as shown in Fig. 3. A square box represents a facility and the input customers are shown by their sequence numbers. In this case competitive ratio is $2|E(\mathcal{M})|$.

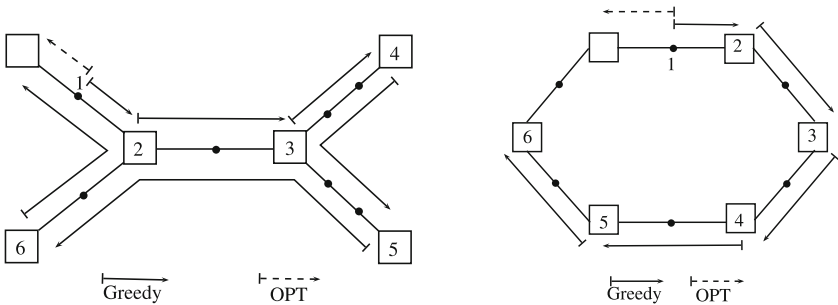


Fig. 3. The configurations of Algorithm Greedy and OPT for a tree and a cycle.

If \mathcal{M} contains a cycle, $\mathcal{R}(\text{Algorithm Greedy})$ does not increase. Consider a set of facilities F placed situated on a cycle. In the worst case $\text{Cost_Greedy}(I)$ is less than $|E(\mathcal{M})|$ and $\text{Cost_OPT}(I)$ is equal to one, as shown in Fig. 3. In this case the competitive ratio is $|E(\mathcal{M})|$.

Now suppose the input sequence I is not well distributed. Let \mathcal{M}' be the minimum subgraph of \mathcal{M} so that all customers are situated on \mathcal{M}' . Consider the set of facilities situated on \mathcal{M}' . In the worst case the customers assigned to those facilities by Algorithm Greedy incur total cost less than $2|E(\mathcal{M}')|$ and OPT incurs only unit cost. If OPT incurs cost x to assign a customer to a remaining facility, then Algorithm Greedy incurs at most $x + |E(\mathcal{M}')|$ cost to assign a customer to that facility. Hence, $\text{Cost_Greedy}(I) \leq \text{Cost_OPT}(I) - 1 + |E(\mathcal{M}')|(|E(\mathcal{M})| - |E(\mathcal{M}')|) + 2|E(\mathcal{M}')|$. It follows that if $|E(\mathcal{M}')|$ is small then Algorithm Greedy will perform similar to OPT. The larger the value of $|E(\mathcal{M}')|$ the more well distributed the input I becomes. Hence $\mathcal{R}(\text{Algorithm Greedy}) \leq 2|E(\mathcal{M})|$. \square

Theorem 4 immediately yields the following corollary.

Corollary 1. *Let \mathcal{M} be a connected unweighted graph and a set of facilities F is placed on the vertices of \mathcal{M} so that distance between two adjacent facilities is equal. Then $\mathcal{R}(\text{Algorithm Greedy}) \leq 4|F|$.*

4.2 Competitive Analysis of Algorithm Optimal-Fill

In Sect. 3.3 we showed that Algorithm Optimal-Fill was more efficient than Algorithm Greedy, when the metric space was a line. In the case of a connected unweighted graph, it is not straight-forward to determine whether Algorithm Optimal-Fill is better than Algorithm Greedy. The answer depends on the number of edges, facilities and the radius of the graph. The following theorem describes the performance of Algorithm Optimal-Fill.

Theorem 5. *Let \mathcal{M} be a connected unweighted graph and a set of facilities F is placed on the vertices of \mathcal{M} . Then $\mathcal{R}(\text{Algorithm Optimal-Fill}) \leq \frac{|E(\mathcal{M})||F|}{r}$.*

Proof. The proof is similar to the analysis of Theorem 4. It is sufficient to consider the case when \mathcal{M} is a tree and I is well distributed. Let x be a vertex in the center of \mathcal{M} which is not a facility. If no such vertex exists, the first customer c_1 is placed on a vertex which is not a facility and the distance from the center of \mathcal{M} is minimum. Otherwise, c_1 is placed on x . In the worst case, Algorithm Optimal-Fill pays a cost equal to the distance between two facilities for each customer, except the first one (see Fig. 4). The adversary pays a cost which is no more than *radius* only for the first customer. Algorithm Optimal-Fill traverses an edge no more than $|F|$ times. Hence, $\mathcal{R}(\text{Algorithm Optimal-Fill})$ is at most $\frac{|E(\mathcal{M})||F|}{r}$. \square

5 Facility Assignment with a Finite Service Time

Until now we have assumed that if a customer c_i is assigned to a facility f_j , then c_i remains there forever. In other words, the service time of an assignment is infinite. Hence a facility with capacity l can provide service to at most l customers. If there are $|F|$ facilities, the total number of customers is limited to $|F|l$. In this section we study the facility assignment problem with a finite service time t . We assume a unit time interval between arrivals of customers. When $t = 1$, the service time is unit. Let c_w be assigned to f_x . and let us consider the case where all facilities have unit capacities ($l = 1$). If c_y is next to c_w then we can also assign c_y to f_x although c_w was assigned to f_x . For unit service time, both Algorithm Greedy and Algorithm Optimal-Fill provide the optimal solution. When the service time is two ($t = 2$), we can not assign c_y to f_x . However, if c_z arrives just after c_y , then we can assign c_z to f_x .

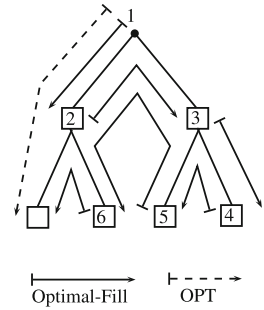


Fig. 4. Worst case of Algorithm Optimal-Fill

Theorem 6. *Let t be the time needed to provide service to a assigned customer. No deterministic algorithm ALG is competitive for $t = 2$.*

Proof. Let $I = (c_1, c_2, \dots, c_n)$ be the input sequence. The adversary places the first customer c_1 between any two adjacent facilities f_i and f_{i+1} . Suppose ALG has assigned c_1 to f_i . The adversary now places c_2, c_3, \dots, c_n exactly on the facilities assigned for c_1, c_2, \dots, c_{n-1} . The adversary runs the optimal algorithm. It assigns c_1 to f_{i+1} , which incurs cost less than d , the distance between f_i and f_{i+1} . The adversary does not pay any cost for the later assignments, because each customer is placed exactly on a facility. However, ALG pays at least d for each assignment except the first one. \square

6 Conclusion

We considered the online facility assignment problem and analyzed several algorithms: Algorithm Greedy, Algorithm σ -Randomized-Greedy and Algorithm Optimal-Fill. We analyzed the performance of these algorithms in two metric spaces: the 1-dimensional line and a simple, connected, unweighted graph. On the line, we made another strong assumption: that the distance between any two adjacent facilities is the same. The algorithms we describe do not generalize to the case when these distances are arbitrary. In Theorem 2, we further assumed that the input sequence of customers is also well distributed. It would be interesting to find out what happens one or both of these assumptions are dropped. In the graph setting we do not make any assumptions about how the facilities are distributed among the vertices, or about how customers are distributed among the vertices. However, our results in this setting are weaker, in the sense that

they depend on parameters such as the number of edges in the graph and its radius. A natural question to ask is whether stronger results exist in the graph setting, as well as in other metric spaces.

References

1. Akagi, T., Nakano, S.: On r -gatherings on the line. In: Wang, J., Yap, C. (eds.) FAW 2015. LNCS, vol. 9130, pp. 25–32. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19647-3_3
2. Antoniadis, A., Barcelo, N., Nugent, M., Pruhs, K., Scquizzato, M.: A $o(n)$ -competitive deterministic algorithm for online matching on a line. In: Bampis, E., Svensson, O. (eds.) WAOA 2014. LNCS, vol. 8952, pp. 11–22. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18263-6_2
3. Armon, A.: On min-max r -gatherings. Theor. Comput. Sci. **412**(7), 573–582 (2011)
4. Bansal, N., Buchbinder, N., Gupta, A., Naor, J.S.: An $O(\log^2 k)$ -competitive algorithm for metric bipartite matching. Algorithmica **68**(2), 390–403 (2012)
5. Bartal, Y., Koutsoupias, E.: On the competitive ratio of the work function algorithm for the k -server problem. Theor. Comput. Sci. **324**(2–3), 337–345 (2004)
6. Bein, W.W., Chrobak, M., Larmore, L.L.: The 3-server problem in the plane. Theor. Comput. Sci. **289**(1), 335–354 (2002)
7. Chrobak, M., Karloff, H., Payne, T., Vishwanathan, S.: New results on server problems. SIAM J. Discrete Math. **4**(2), 291–300 (1990)
8. Chrobak, M., Larmore, L.L.: An optimal on-line algorithm for k -servers on trees. SIAM J. Comput. **20**(1), 144–148 (1991)
9. Salhi, S., Drezner, E.: Facility location: a survey of applications and methods. J. Oper. Research Soc. **47**(11), 1421 (1995). <https://doi.org/10.2307/3010210>
10. Drezner, Z., Hamacher, H.W.: Facility Location: Applications and Theory. Springer, Heidelberg (2004)
11. Kalyanasundaram, B., Pruhs, K.: Online weighted matching. J. Algorithms **14**(3), 478–488 (1993)
12. Kao, M.Y., Reif, J.H., Tate, S.R.: Searching in an unknown environment: an optimal randomized algorithm for the cow-path problem. Inf. Comput. **131**(1), 63–79 (1996)
13. Khuller, S., Mitchell, S.G., Vazirani, V.V.: On-line algorithms for weighted bipartite matching and stable marriages. Theor. Comput. Sci. **127**(2), 255–267 (1994)
14. Kleinberg, J.M.: A lower bound for two-server balancing algorithms. Inf. Process. Lett. **52**(1), 39–43 (1994)
15. Koutsoupias, E., Papadimitriou, C.: The 2-evader problem. Inf. Process. Lett. **57**(5), 249–252 (1996). [https://doi.org/10.1016/0020-0190\(96\)00010-5](https://doi.org/10.1016/0020-0190(96)00010-5)
16. Manasse, M.S., McGeoch, L.A., Sleator, D.D.: Competitive algorithms for server problems. J. Algorithms **11**(2), 208–230 (1990)
17. Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency, Algorithms and Combinatorics, vol. 24. Springer, Heidelberg (2003)
18. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**(2), 202–208 (1985)

Fault-Tolerant Complete Visibility for Asynchronous Robots with Lights Under One-Axis Agreement

Aisha Aljohani, Pavan Poudel, and Gokarna Sharma^(✉) 

Department of Computer Science, Kent State University, Kent, OH 44242, USA
aaljoha6@kent.edu, {ppoudel,sharma}@cs.kent.edu

Abstract. We consider the distributed setting of N autonomous mobile robots that operate in *Look-Compute-Move* (LCM) cycles and communicate with other robots using colored lights under the *robots with lights* model. We study the fundamental COMPLETE VISIBILITY problem of repositioning N robots on a plane so that each robot is visible to all others. We assume *obstructed visibility* under which a robot cannot see another robot if a third robot is positioned between them on the straight line connecting them. We are interested in *fault-tolerant* algorithms. We study fault-tolerance with respect to failures on the mobility of robots. Therefore, any algorithm for COMPLETE VISIBILITY is required to provide visibility between all non-faulty robots, independently of the behavior of the faulty ones. We model mobility failures as *crash faults* in which each faulty robot is allowed to stop its movement at any time and, once the faulty robot stopped moving, that robot will remain stationary indefinitely thereafter. There exists an algorithm for this problem that tolerates a single faulty robot in the semi-synchronous setting under both-axis agreement. In this paper, we provide the first algorithm for this problem that tolerates $f \leq N$ faulty robots in the asynchronous setting under one-axis agreement. The proposed algorithm is *collision-free* – robots do not share positions and their paths do not cross, *energy efficient* – each robot performs at most one move, and handles *non-rigidity* of the robot movements.

1 Introduction

In the well-celebrated classical model of distributed computing by mobile robots, each robot is modeled as a point in the plane [11]. The robots are assumed to be *autonomous* (no external control), *anonymous* (no unique identifiers), *indistinguishable* (no external identifiers), and *disoriented* (no agreement on local coordinate systems and units of distance measures). They execute the same algorithm. Each robot proceeds in *Look-Compute-Move* (LCM) cycles: When a robot becomes active, it first gets a snapshot of its surroundings (*Look*), then computes a destination point based on the snapshot (*Compute*), and finally moves towards the destination point (*Move*). Moreover, the robots are *oblivious*, i.e., in each

LCM cycle, each robot has no memory of its past LCM cycles [11]. Furthermore, the robots are *silent* because they do not communicate directly, and only vision and mobility enable them to coordinate their actions.

While silence has advantages, direct communication is preferred in many other situations, for example, hostile environments, which makes coordination efficient and relatively viable. One model that incorporates direct communication is the *robots with lights* model [9, 11, 15], where each robot has an externally visible light that can assume colors from a constant sized set, and hence robots can explicitly communicate with each other using these colors. The colors are *persistent*; i.e., the color is not erased at the end of a cycle. Except for lights, the robots are oblivious as in the classical model.

Di Luna *et al.* [13] gave the first algorithm for robots with lights to solve the fundamental COMPLETE VISIBILITY problem defined as follows: Given an arbitrary initial configuration of N autonomous mobile robots located in distinct points on a plane, they reach a configuration in which each robot is in a distinct position from which it can see all other robots. Initially, some robots may be obstructed from the view of other robots and the total number of robots, N , is not known to robots. The importance of this problem is that it makes it possible to solve many other robotic problems, including gathering, shape formation, and leader election, under obstructed visibility [12, 16]. Most importantly, it recovers unobstructed visibility configuration starting from an obstructed visibility configuration. Subsequently, several papers [12, 16] solved this problem minimizing number of colors. Recently, faster runtime algorithms [18–20] were studied for this problem in the lights model (details in **Related Work**). This problem is also called MUTUAL VISIBILITY in some papers [12, 16].

In this paper, we are interested in the fault-tolerant algorithms for COMPLETE VISIBILITY in the robots with lights model. We study fault-tolerance with respect to failures on the mobility of robots. Therefore, any algorithm for COMPLETE VISIBILITY is required to provide visibility between all non-faulty robots, independently of the behavior of the faulty ones and the locations of the faulty robots. We model mobility failures as *crash faults* where each faulty robot is allowed to stop its movement at any moment of time and remains stationary indefinitely thereafter [2]. The only previous work that studied faults for this problem is [3] in which the authors solved the problem for a single faulty robot in the semi-synchronous setting under both-axis agreement. In this paper, we focus on solving this problem tolerating $f > 1$ faulty robots in the weakest fully asynchronous setting and under weaker one-axis agreement.

Contributions. We consider the same robot model as in [12, 13], namely, robots are oblivious except for a persistent light that can assume a constant number of colors. Visibility could be obstructed by other robots in the line of sight and N is not known. We assume that the setting is *asynchronous* where there is no notion of common time and robots perform their LCM cycles at arbitrary time. We also assume *non-rigid* moves – a robot in motion can be stopped (by an adversary) before it reaches its destination point with the only constraint that the robot moves at least distance $\Delta > 0$, otherwise COMPLETE VISIBILITY

cannot be solved [12]. As in [13], we assume that two robots cannot head to the same destination and their paths when they move cannot cross. This would constitute a *collision*. Furthermore, we assume *one-axis agreement* – all robots agree on either x -axis or y -axis [11]. In this paper, we prove the following result.

Theorem 1. *For any input configuration of $N \geq 3$ robots (with lights) in distinct positions in a plane, COMPLETE VISIBILITY can be solved tolerating (up to) N crash-faulty robots using 4 colors in $\mathcal{O}(N)$ time without collisions in the asynchronous setting.*

To the best of our knowledge, Theorem 1 is the first result for COMPLETE VISIBILITY that tolerates (up to) N faulty robots in the asynchronous setting. In the semi-synchronous (and also fully synchronous) setting, Theorem 1 only needs 2 colors, which is optimal with respect to the number of colors used [12]. One prominent feature of our algorithm is that each robot moves at most once during the execution and it has implications on energy efficiency of robots on solving COMPLETE VISIBILITY.

When the robots are fault-free, the idea used in the existing algorithms [12, 13, 16, 18–20] is to reposition the robots so that they all become corners of a N -corner convex hull. After that, a property of the convex hull guarantees that there is a line connecting each corner with all others of the hull without any third robot being collinear on those lines, i.e., a convex hull naturally solves COMPLETE VISIBILITY. However, when robots are faulty, the faulty robots may be in the interior of the convex hull and it is challenging to guarantee that all non-faulty robots see each other (that is, faulty robots do not block the view for the non-faulty robots to see each other). Since robots are oblivious and non-faulty robots do not know which robots are faulty, this task becomes quite challenging. Aljohani and Sharma [3] managed to address this challenge only when at most one robot in the interior of the hull experiences fault. In this paper, we develop a technique in which non-faulty robots do not need to be positioned on the corners of a hull to see other non-faulty robots and this gives the scalability on number of faults that can be tolerated. Our idea is to move the robots in a sequence starting from the Southmost robot and ending at the Northmost robot, and guarantee that, when a robot makes a move, it moves to a position in such a way that it sees from that position all robots that are positioned South of it (both faulty and non-faulty).

Related Work. Di Luna *et al.* [13] gave the first algorithm for COMPLETE VISIBILITY in the robots with lights model. They solved the problem using 6 colors in the semi-synchronous setting and 10 colors in the asynchronous setting under both rigid and non-rigid movements. Di Luna *et al.* [12] solved the problem using 2 colors in the semi-synchronous setting under rigid movements. They solved the problem using 3 colors in the semi-synchronous setting under non-rigid movements and in the asynchronous setting under rigid movements. They also provided a solution using 3 colors in the asynchronous setting under non-rigid movements under one-axis agreement. Sharma *et al.* [16] improved the number of colors in the solution of Di Luna *et al.* [12] from 3 to 2. In the classical

oblivious model (with no lights), Bhagat *et al.* [5] solved COMPLETE VISIBILITY under one-axis agreement without the need of robots to be positioned on the corners of a convex hull. However, all these results provided no runtime analysis. Moreover, none of these results tolerate faults.

Vaidyanathan *et al.* [20] considered runtime for the very first time for COMPLETE VISIBILITY giving an algorithm that runs in $\mathcal{O}(\log N)$ time using $\mathcal{O}(1)$ colors in the fully synchronous setting under rigid movements. Later, Sharma *et al.* [18] provided an $\mathcal{O}(1)$ time algorithm using $\mathcal{O}(1)$ colors in the semi-synchronous setting under rigid movements. Recently, Sharma *et al.* [17, 19] provided an $\mathcal{O}(1)$ time algorithm using $\mathcal{O}(1)$ colors in the asynchronous setting under rigid movements. However, all these algorithms are not fault-tolerant. Aljohani and Sharma [3] provided an algorithm that tolerates one faulty robot when robots have both axis agreement in the semi-synchronous setting under rigid movements. The algorithm we present in this paper assumes one-axis agreement, handles non-rigid movements, and works in the fully asynchronous setting.

The computational power of the robots with lights model is studied in [9] while the robots are working on the Euclidean plane and in [10] while the robots are working on graphs.

The obstructed visibility, in general, is considered in the problem of uniformly spreading robots operating on a line [6] and also in the near-gathering problem [14] where collisions must be avoided among robots. It is also considered in the so-called *fat robots* model [1, 8] in which robots are not points, but non-transparent unit discs. However, these works do not consider faulty robots. The faults are considered for the gathering problem in the classical oblivious robots model [2, 4]. Our definition of crash faults is borrowed from [2].

Paper Organization. The rest of the paper is organized as follows. We present the robot model and preliminaries in Sect. 2. We then present and analyze our fault-tolerant COMPLETE VISIBILITY algorithm in Sect. 3 and conclude in Sect. 4. Some proofs and pseudocodes are omitted due to space constraints.

2 Model and Preliminaries

Robots. We consider a distributed system of N autonomous robots from a set $\mathcal{Q} = \{r_1, \dots, r_N\}$. Each robot $r_i \in \mathcal{Q}$ is a (dimensionless) point that can move in the two-dimensional Euclidean plane \mathbb{R}^2 . Throughout the paper, we denote by r_i the robot r_i as well as its position p_i in \mathbb{R}^2 . We assume that each robot $r_i \in \mathcal{Q}$ shares one coordinate axis with other robots in \mathcal{Q} , i.e., they agree on either x -axis or y -axis (we use y -axis).

A robot r_i can see, and be visible to, another robot r_j if and only if there is no third robot r_k in the line segment $\overline{r_i r_j}$ connecting r_i and r_j . Each robot $r_i \in \mathcal{Q}$ has a light that can assume a color at a time from a set of constant number of different colors. We denote the color of a robot $r_i \in \mathcal{Q}$ at any time by variable $r_i.light$. If $r_i.light = \mathbf{Red}$, then it means that r_i has color \mathbf{Red} . Moreover, the color \mathbf{Red} of r_i is seen by all robots that can see r_i at that time (r_i also can see its current color). The execution starts at time $t = 0$ and at time $t = 0$ all robots in \mathcal{Q} are stationary with each of them colored \mathbf{Off} .

Look-Compute-Move. Each robot r_i is either active or inactive. When a robot r_i becomes active, it performs the “Look-Compute-Move” cycle as described below.

- *Look:* For each robot r_j that is visible to it, r_i can observe the position of r_j on the plane and the color of the light of r_j . Robot r_i can also observe its own color and position; that is, r_i is visible to itself. Each robot observes positions on its own frame of reference. That is, two different robots observing the position of the same point may produce different coordinates. However, a robot observes the positions of points accurately within its own reference frame.
- *Compute:* In any LCM cycle, r_i may perform an arbitrary computation using only the colors and positions observed during the “look” portion of that LCM cycle. This includes determination of a (possibly) new position and color for r_i for the start of next LCM cycle. Robot r_i maintains this new color from that cycle to the next.
- *Move:* At the end of the LCM cycle, r_i changes its light to the new color and moves to its new position.

Robot Activation. In the fully synchronous setting ($\mathcal{FSYN}\mathcal{C}$), every robot is active in every LCM cycle. In the semi-synchronous setting ($\mathcal{SSYN}\mathcal{C}$), at least one robot is active, and over an infinite number of LCM cycles, every robot is active infinitely often. In the asynchronous setting ($\mathcal{ASYN}\mathcal{C}$), there is no common notion of time and no assumption is made on the number and frequency of LCM cycles in which a robot can be active. The only guarantee is that every robot is active infinitely often. The moves of the robots may be *non-rigid* – during the *Move* phase the robots move in a straight line but they may stop their movement before they reach to the destination point computed in the *Compute* phase, with the only exception that they move at least some distance $\Delta > 0$. We assume that the faulty robot can crash at any moment of time. After the robot crashes, it does not move again (i.e., stays stationary indefinitely). However, even after the robot crashes, we assume that it does not have impact on the operations of its light. That is, the robot can correctly change its color to any color in the color set according to the algorithm. We will argue in Sect. 4 that it seems necessary to guarantee termination tolerating $f > 1$ robot faults even under one-axis agreement.

Runtime. For the $\mathcal{FSYN}\mathcal{C}$ model, we measure time in rounds, where one round is one LCM cycle. As a robot in the $\mathcal{SSYN}\mathcal{C}$ (and $\mathcal{ASYN}\mathcal{C}$) model could stay inactive for an indeterminate number of cycles and (time), we use the notion of an epoch to measure runtime [7]. Let t_0 denote the start time of the computation. Epoch i is time period from t_{i-1} to t_i where t_i is the earliest time after t_{i-1} when all robots have executed a complete LCM cycle at least once. In the $\mathcal{FSYN}\mathcal{C}$ model, an epoch is one round (one LCM cycle). We will use the term “time” generically to mean rounds for the $\mathcal{FSYN}\mathcal{C}$ model and epochs for the $\mathcal{SSYN}\mathcal{C}$ and $\mathcal{ASYN}\mathcal{C}$ models.

Configuration. A configuration $\mathbf{C}_t = \{(r_1^t, col_1^t), \dots, (r_N^t, col_N^t)\}$ defines the positions of the robots in \mathcal{Q} and their colors for any time $t \geq 0$. A configuration for a robot $r_i \in \mathcal{Q}$, $\mathbf{C}_t(r_i)$, defines the positions of the robots in \mathcal{Q} that are visible to r_i (including r_i) and their colors, i.e., $\mathbf{C}_t(r_i) \subseteq \mathbf{C}_t$, at time t . For simplicity, we sometime write $\mathbf{C}, \mathbf{C}(r_i)$ to denote $\mathbf{C}_t, \mathbf{C}_t(r_i)$, respectively.

Visible Area. Let \mathcal{A} be a set of points (which are the current positions of the robots in \mathbb{R}^2) and \mathbf{P} be the convex hull of the points in \mathcal{A} . \mathbf{P} has the property that all the points of \mathcal{A} are either in the perimeter or in its interior. The points in the perimeter of \mathbf{P} are either on corners of \mathbf{P} or on the edges of \mathbf{P} , which we call corner and side points of \mathbf{P} , respectively. Let $\mathcal{Q}_c, \mathcal{Q}_s, \mathcal{Q}_i$ be the set of points at corners, sides, and the interior of \mathbf{P} . Moreover, let c_i be a corner point of \mathbf{P} and a, b be the counterclockwise and clockwise neighbors of c_i in the perimeter of \mathbf{P} . The *visible area* for c_i , denoted as $Visible_Area(c_i)$, is a polygonal subregion inside \mathbf{P} within the triangle c_iuw , where u, w are the midpoints of edges $\overline{c_i a}, \overline{c_i b}$, respectively. According to this computation, the visible areas for any two corner points of \mathbf{P} are disjoint. Due to obstructed visibility, $Visible_Area(c_i)$ is computed based on $\mathbf{C}(c_i)$ and the corresponding hull $\mathbf{P}(c_i)$. This computation is used in Sect. 3.

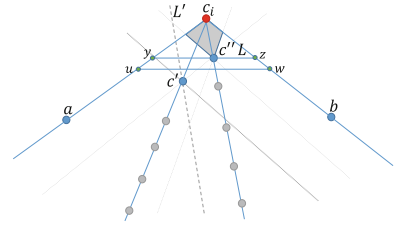


Fig. 1. Visible area

We now outline how $Visible_Area(c_i)$ is computed for any corner c_i of \mathbf{P} . The pseudocode is omitted due to space constraints. Initially, c_i sets the triangle c_iuw as its $Visible_Area(c_i)$. However, if c_i sees some point of \mathcal{A} inside c_iuw , then it sets as $Visible_Area(c_i)$ the triangle c_iyz such that there is no point inside c_iyz . Note that \overline{yz} is parallel to \overline{ab} . Let c' be a point in $\mathbf{C}(c_i)$. For every other point $c'' \in \mathbf{C}(c_i), c'' \neq c', c'' \neq c_i$, c_i computes a line, L' , parallel to $\overline{c_i c''}$ passing through c' . Let HP be the half-plane divided by L' such that c_i is in HP . Corner c_i then updates its $Visible_Area(c_i)$ by keeping only the portion of $Visible_Area(c_i)$ that is in the half-plane HP . This process is repeated for all $c' \in \mathbf{C}(c_i) \setminus \{c_i\}$ and $Visible_Area(c_i)$ is updated in every iteration. Now from the area $Visible_Area(c_i)$ that remains, c_i removes the points that are in the perimeter of $Visible_Area(c_i)$ and also the points that are part of the lines $\overline{c_i x}, x \in \mathbf{C}(c_i) \setminus \{a, b, c_i\}$, passing inside of $Visible_Area(c_i)$. This removal of points is crucial to guarantee that when c_i moves to a point in $Visible_Area(c_i)$, it does not become collinear with any robot in $\mathcal{Q}_s, \mathcal{Q}_i$. Figure 1 illustrates the computation of $Visible_Area(c_i)$; the shaded area is $Visible_Area(c_i)$ for corner c_i of \mathbf{P} except the points on the lines inside it (e.g., the point of lines $\overline{c_i c'}$ and $\overline{c_i c''}$ inside $Visible_Area(c_i)$). We have the following lemma from [18].

Lemma 1. *Visible_Area(c_i) for each corner robot c_i in \mathbf{P} is non-empty. Moreover, when c_i moves to a point inside $Visible_Area(c_i)$ and no other robot in \mathbf{P} is moving simultaneously with c_i , then c_i remains as a corner of \mathbf{P} and all the other robots in \mathbf{P} are visible to c_i (and vice-versa).*

3 Algorithm

In this section, we present our COMPLETE VISIBILITY algorithm for $N \geq 3$ robots with lights tolerating $f \leq N$ faulty robots, starting from any arbitrary initial configuration with robots being in the distinct positions in a plane. The algorithm works in the \mathcal{ASYNC} setting handling non-rigid moves, under the assumption that robots have one-axis agreement. We first provide a high level overview and then give its details.

Algorithm 1. COMPLETE VISIBILITY for a robot $r_i \in \mathcal{Q}$ in the \mathcal{ASYNC} model

```

1 // Look-Compute-Move cycle for each robot  $r_i \in \mathcal{Q}$ 
2  $Hor(r_i) \leftarrow$  horizontal line passing through  $r_i$ ;
3  $\mathbf{C}(r_i) \leftarrow$  configuration  $\mathbf{C}$  for robot  $r_i$  (including  $r_i$ );
4  $\mathbf{C}_{Hor}(r_i) \leftarrow$  configuration  $\mathbf{C}(r_i)$  of robots South of  $Hor(r_i)$ ;
5 if  $|\mathbf{C}_{Hor}(r_i)| = \emptyset$  then
6     if  $r_i.light = \text{Off} \wedge$  there is no other robot on  $Hor(r_i)$  then  $r_i.light = \text{Final}$ ;
7     if  $r_i.light = \text{Off} \wedge$  there are robots on  $Hor(r_i) \wedge r_i$  is the endpoint robot on
8          $Hor(r_i)$  then  $r_i.light = \text{Intermediate}$ ;
9     if  $r_i.light = \text{Intermediate}$  then
10         Set  $r_i.light = \text{Transit}$  and move vertically South distance 1;
11     if  $r_i.light = \text{Transit} \wedge r_i$  sees no Intermediate colored robot then
12          $r_i.light = \text{Final}$ ;
13 if  $|\mathbf{C}_{Hor}(r_i)| \neq \emptyset$  then
14     if  $r_i.light = \text{Off} \wedge$  there is no other robot on  $Hor(r_i) \wedge r_i$  sees no robot
15         colored Off, Intermediate, or Transit South of  $Hor(r_i)$  then
16          $V_i \leftarrow \text{Visible\_Area}(r_i, \mathbf{C}_{Hor}(r_i) \cup \{r_i\})$ ;
17          $Hor(r_j) \leftarrow$  horizontal line passing through robot  $r_j$  South of  $Hor(r_i)$ 
18             closest to  $Hor(r_i)$ ;
19          $V_i \leftarrow V_i$  after removing the area South of  $Hor(r_j)$ ;
20         Set  $r_i.light = \text{Transit}$  and move to a point in  $V_i$ ;
21     if  $r_i.light = \text{Off} \wedge$  there are robots on  $Hor(r_i) \wedge r_i$  is the endpoint robot on
22          $Hor(r_i) \wedge$  there is no robot colored Off, Intermediate, or Transit South of
23          $Hor(r_i)$  then  $r_i.light = \text{Intermediate}$ ;
24     if  $r_i.light = \text{Intermediate}$  then
25          $V_i \leftarrow \text{Visible\_Area}(r_i, \mathbf{C}_{Hor}(r_i) \cup \{r_i\})$ ;
26         if there is another robot  $r_k$  on  $Hor(r_i)$  then
27              $L' \leftarrow$  line connecting  $r_k$  with a robot  $r$  South of  $Hor(r_i)$  such that
28                 there is no robot in the cone area formed by lines  $Hor(r_i)$  and  $\overrightarrow{r_k r}$ ;
29              $Hor(r_j) \leftarrow$  horizontal line passing through robot  $r_j$  South of
30                  $Hor(r_i)$  closest to  $Hor(r_i)$ ;
31              $V_i \leftarrow V_i$  after removing the area beyond line  $L'$  and South of
32                  $Hor(r_j)$ ;
33         else  $V_i \leftarrow V_i$  after removing the area South of  $Hor(r_j)$ ;
34         Set  $r_i.light = \text{Transit}$  and move to a point in  $V_i$ ;
35     if  $r_i.light = \text{Transit} \wedge r_i$  sees no Intermediate colored robot then
36          $r_i.light = \text{Final}$ ;

```

High Level Overview of the Algorithm. The goal is to make robots progress toward a configuration where no three non-faulty robots are collinear and no faulty robot is in a line connecting two non-faulty robots. When all (non-faulty) robots in \mathcal{Q} satisfy this property, this solves COMPLETE VISIBILITY. All previous algorithms for COMPLETE VISIBILITY [12, 13, 16, 18–20] arrange robots on the corners of a convex hull. Although convex hull is not the required condition for COMPLETE VISIBILITY (i.e., it is a sufficient condition), the correctness analysis becomes easier. However, when faulty robots are in hull’s interior, it is difficult to arrange robots on the corners of a hull.

Our idea is to develop a technique which does not require robots to be positioned on the corners of a convex hull, and hence scales on the number of faults it can tolerate. Let \mathbf{C}_0 be any initial configuration of the robots in \mathcal{Q} with robots being in the distinct positions on the plane. Let L be a vertical line (robots agree on y -axis). The robots in \mathcal{Q} can be projected to L so that all the robots are between positions b_L and t_L on L , where b_L is bottommost position on L that the robots in \mathcal{Q} are projected to and t_L is the topmost position on L that the robots on L are projected to. There can be at most N different points on L that the robots in \mathcal{Q} can be projected to. The idea in our algorithm is to ask the robots whose positions were projected on b_L to move first. Those robots then terminate. Until this time, the robots that are not projected to b_L do nothing. After that, the robots that are projected to a point b_L^1 (the neighboring point of b_L on L) move and terminate; the robots that are not projected to b_L^1 do nothing. This process then repeats until the robots that are projected to t_L move and terminate. The algorithm then finishes. We show that this process guarantees that COMPLETE VISIBILITY is achieved for the non-faulty robots in \mathcal{Q} even when (up to) $f \leq N$ robots experience faults.

At any time which robots of \mathcal{Q} move and which robots of \mathcal{Q} do not move is determined through the colors displayed on the lights. We need to be careful how the robots move when two or more robots are projected to the same position on L . We handle this issue by asking robots that are at the two extremal points on the horizontal line they are positioned on to move first and then their neighbors can move subsequently.

In \mathbf{C}_0 , all robots in \mathcal{Q} have color `Off` and are stationary. But, in the COMPLETE VISIBILITY configuration, all robots in \mathcal{Q} have color `Final`. The algorithm uses four colors `Final`, `Transit`, `Intermediate`, and `Off`. The colors `Intermediate` and `Transit` are to synchronize the simultaneous moves of the (at most) two robots at any moment of time in the $\mathcal{ASYN}\mathcal{C}$ setting to make sure that COMPLETE VISIBILITY is achieved satisfying Theorem 1. These two colors are not required in the $\mathcal{SSYN}\mathcal{C}$ (and $\mathcal{FSYN}\mathcal{C}$) setting (details in Sect. 4). Moreover, robots do not know N and their termination decision is solely based on the color they assume. The robots work autonomously only having the information about the robots they see.

Details of the Algorithm. The pseudocode of the algorithm is given in Algorithm 1. Initially at \mathbf{C}_0 , the lights of all robots are set to color `Off` and the robots are stationary. Let r_i be a robot in \mathcal{Q} . Let $Hor(r_i)$ be a horizontal line passing

through the position of r_i . We first discuss how r_i moves if it sees no robot South of $Hor(r_i)$, i.e., r_i is the Southmost robot in the configuration. Robot r_i can determine whether it is a Southmost robot or not as it knows y -axis. Robot r_i simply changes its color to **Final** without moving if it sees no other robot on $Hor(r_i)$. If r_i sees some other robot on $Hor(r_i)$, it changes its color to **Intermediate** (without moving) if it is positioned on $Hor(r_i)$ such that it sees robots on only one side on $Hor(r_i)$. We call r_i the *endpoint* robot on $Hor(r_i)$ if the above condition is satisfied. Otherwise, r_i is on $Hor(r_i)$ with at least a robot on $Hor(r_i)$ on its both sides and r_i does nothing until it either becomes the endpoint robot on $Hor(r_i)$ (fault-free case) or the robot on at least one side of $Hor(r_i)$ is colored **Final** (faulty case). After r_i is colored **Intermediate**, it assumes color **Transit** and moves distance 1 vertically South. If r_i is colored **Transit**, it assumes color **Final** if it sees no **Intermediate** colored robot. If r_i sees an **Intermediate** colored robot, they both were on $Hor(r_i)$ before r_i moved, and the waiting makes sure that the **Intermediate** colored robot also moves before r_i terminates. This makes synchronization easier in the *ASYNC* setting. We will discuss in Sect. 4 this is not needed in the *SSYNC* (and *FSYNC*) setting.

We now discuss how r_i moves if it sees at least a robot South of $Hor(r_i)$. Robot r_i does not move until it sees at least a **Off**, **Intermediate**, or **Transit** colored robot South of $Hor(r_i)$. The idea here is for r_i to compute $Visible_Area(r_i, \mathbf{C}_{Hor(r_i)} \cup \{r_i\})$ considering the robots South of $Hor(r_i)$ that it sees and move to a point in $Visible_Area(r_i, \mathbf{C}_{Hor(r_i)} \cup \{r_i\})$. If r_i is the only robot on $Hor(r_i)$, it assumes color **Transit** and moves to a point in $Visible_Area(r_i, \mathbf{C}_{Hor(r_i)} \cup \{r_i\})$. If r_i is not the only robot on $Hor(r_i)$ but an endpoint robot on $Hor(r_i)$, then it first assumes color **Intermediate** from **Off**. After r_i colored **Intermediate**, it moves as follows: r_i computes $Visible_Area(r_i, \mathbf{C}_{Hor(r_i)} \cup \{r_i\})$ and moves to a point in $Visible_Area(r_i, \mathbf{C}_{Hor(r_i)} \cup \{r_i\})$ assuming color **Transit**. After colored **Transit**, it sets its light to **Final** if it does not see any **Intermediate** colored robot. If it sees an **Intermediate** colored robot r_j , r_j must be North of $Hor(r_i)$ and it waits until r_j assumes color **Transit**. After colored **Final**, r_i terminates its computation when it becomes active next time.

We restrict how a point in $Visible_Area(r_i, \mathbf{C}_{Hor(r_i)} \cup \{r_i\})$ is selected to avoid robot collisions. Suppose $Hor(r_j)$ is the horizontal line passing through a robot r_j South of $Hor(r_i)$ such that there is no robot between lines $Hor(r_i)$ and $Hor(r_j)$. We restrict that r_i can not move to positions of $Hor(r_j)$ or South of it. This will avoid collisions between robots of $Hor(r_i)$ and $Hor(r_j)$. To avoid collisions between two robots of $Hor(r_i)$ (that can move simultaneously) and also to make sure that the moves of those robots do not block the visibility of each other to see the robots South of $Hor(r_i)$, we restrict r_i not to move on or beyond $\bar{r}_k\bar{r}$ (in addition to not moving beyond $Hor(r_j)$), where r_k is the neighboring robot of r_i on $Hor(r_i)$ and r is the robot South of $Hor(r_i)$ such that there is no robot in the *cone area* formed by lines $Hor(r_i)$ and $\bar{r}_k\bar{r}$. Figure 2 illustrates these ideas.

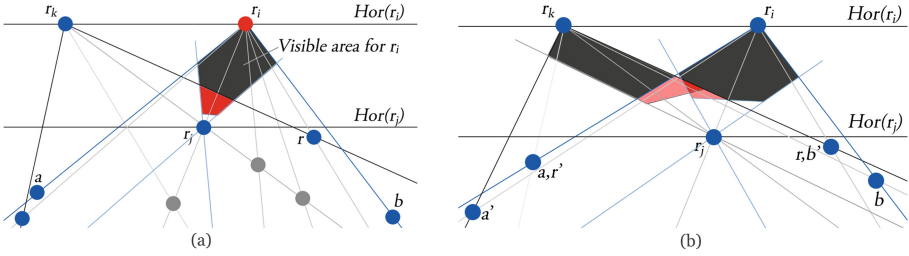


Fig. 2. (a) $Visible_area(r_i)$ for r_i (black region) is computed by removing the part of it beyond $\overline{r_k r}$ (red region) and (b) disjoint $Visible_Area(r_i)$ and $Visible_Area(r_k)$ for two robots r_i, r_k on $Hor(r_i)$ (black regions) using the technique of (a). (Color figure online)

Analysis of the Algorithm. We now analyze the correctness of the algorithm. Particularly, we show that the algorithm solves COMPLETE VISIBILITY starting from any initial configuration \mathbf{C}_0 with all robots in \mathcal{Q} being in the distinct positions in the plane and (up to) N robots become faulty. We further show that the algorithm terminates in $\mathcal{O}(N)$ time and the execution is collision- and deadlock-free. We start with the following lemma.

Lemma 2. *Let $Hor(r_i)$ and $Hor(r_j)$ be horizontal lines passing through robots r_i, r_j such that there is no robot in the area between lines $Hor(r_i)$ and $Hor(r_j)$ and $Hor(r_j)$ is in South of $Hor(r_i)$. No robot on $Hor(r_i)$ is colored Intermediate, Transit, or Final until all the robots on $Hor(r_j)$ are colored Final.*

Lemma 3. *When a robot r_i on $Hor(r_i)$ computes $Visible_Area(r_i, \mathbf{C}_{Hor(r_i)} \cup \{r_i\})$, it is a corner of the convex hull \mathbf{P} of the robots of $\mathbf{C}_{Hor(r_i)} \cup \{r_i\}$.*

Proof. We have that $\mathbf{C}_{Hor(r_i)}$ consists of the robots South of $Hor(r_i)$ that r_i sees. Therefore, when r_i computes a convex hull $\mathbf{P}(r_i)$ of the robots in $\mathbf{C}_{Hor(r_i)} \cup \{r_i\}$, it makes an angle of $<180^\circ$ with its two neighboring corners of $\mathbf{P}(r_i)$ since all the robots on $\mathbf{C}_{Hor(r_i)}$ are in one side of $Hor(r_i)$. \square

Lemma 4. *When a (non-faulty) robot r_i moves once, it sees all robots (both faulty and non-faulty) South of $Hor(r_i)$.*

Proof. We have from Lemma 1 that when a corner r_i of a convex hull \mathbf{P} moves to a point in $Visible_Area(r_i, \mathbf{C}_{Hor(r_i)} \cup \{r_i\})$ and no other robot is moving simultaneously with r_i , r_i sees all other robots of \mathbf{P} (corners, side, and interior). We have from Lemma 3 that r_i is a corner of a convex hull \mathbf{P} formed by the robots in $\mathbf{C}_{Hor(r_i)} \cup \{r_i\}$. When r_i moves in Algorithm 1, no other robot of \mathbf{P} formed from $\mathbf{C}_{Hor(r_i)} \cup \{r_i\}$ is moving, therefore r_i sees all the robots that are South of $Hor(r_i)$. It only remains to show that, at most one other robot r on $Hor(r_i)$ that is moving simultaneously with r_i is also visible to r_i and vice-versa. This is immediate from the visible areas $Visible_Area(r_i, \mathbf{C}_{Hor(r_i)} \cup \{r_i\})$ and $Visible_Area(r, \mathbf{C}_{Hor(r)} \cup \{r\})$ computed by r_i and r , respectively. Let a, b be

the left and right neighbors of r_i in $\mathbf{P}(r_i)$ among the robots in $\mathbf{C}_{Hor}(r_i) \cup \{r_i\}$. Moreover, let a', b' be the left and right neighbors of r in $\mathbf{P}(r)$ among the robots in $\mathbf{C}_{Hor}(r) \cup \{r\}$. We have that $Visible_Area(r_i)$ does not contain the area beyond line $\overline{r'b'}$ and $Visible_Area(r)$ does not contain the area beyond line $\overline{r_i a}$ (Fig. 2). Therefore, even if r_i, r move simultaneously, r_i does not become collinear with r in line $\overline{r x}$ connecting r with any robot $x \in \mathbf{C}_{Hor}(r)$ and r does not become collinear with r_i in line $\overline{r_i x}$ connecting r_i with any robot $x \in \mathbf{C}_{Hor}(r_i)$. Moreover, even after r_i and r move simultaneously, r_i sees r and vice-versa follows immediately since they move in the area between $Hor(r_i)$ and $Hor(r_j)$ with r_j the same robot in the view of both r_i, r and there is no third robot in that area. \square

Lemma 5. *Each (non-faulty) robot does at most one move during entire execution.*

Proof. Pick any robot r_i . If it is a Southmost robot and there is no robot on $Hor(r_i)$, it terminates without moving. If it picks color **Intermediate**, then it does so without moving. If it picks color **Transit**, then it moves. If r_i is already colored **Transit**, it changes its color to **Final** without moving. If r_i is colored **Final**, it terminates. Therefore, a robot r_i moves only once when it picks color **Transit** either from **Off** or from **Intermediate**. Therefore, each non-faulty robot moves at most once. \square

Lemma 6. *Algorithm 1 is collision-free.*

Proof. Let $Hor(r_i)$ and $Hor(r_j)$ be two horizontal lines such that there is no robot in the area between $Hor(r_i)$ and $Hor(r_j)$. Let $Hor(r_j)$ be South of $Hor(r_i)$. The robots on lines $Hor(r_i)$ and $Hor(r_j)$ do not collide since the robots on $Hor(r_i)$ never reach to positions of $Hor(r_j)$ and the robots on $Hor(r_j)$ never move North of $Hor(r_j)$. Therefore, it only remains to show that the robots on $Hor(r_i)$ do not collide with each other. We have that at most 2 endpoint robots r_i, r_k on $Hor(r_i)$ move simultaneously. The robots move in such a way that r_i does not cross line $\overline{r_k r}$ and r_k does not cross line $\overline{r_i a}$ (as defined in Fig. 2b) and hence this avoids collisions between them. \square

Lemma 7. *Algorithm 1 is deadlock-free.*

Lemma 8. *Algorithm 1 runs for $\mathcal{O}(N)$ epochs.*

Lemma 9. *The non-rigid movements of robots do not impact the guarantees of the algorithm.*

Proof. Let d_i be the point in $Visible_Area(r_i)$ that r_i moves under rigid movements. Let $\overline{r_i d_i}$ be the line segment connecting r_i with d_i before r_i moves to d_i . Under non-rigid movements, r_i may stop anywhere between r_i and d_i (we know that it does not stop at r_i since it moves at least $\Delta > 0$). We have from $Visible_Area(r_i)$ that r_i is visible to all other non-moving robots if it moves to any point in $Visible_Area(r_i)$. According to the visible area construction,

all points in line $\overline{r_i d_i}$ contain inside the visible area $Visible_Area(r_i)$. Therefore, even under non-rigid movements, the algorithm provides all guarantees we obtained under rigid movements. \square

Proof of Theorem 1. We have Theorem 1 combining the results of Lemmas 4–9.

4 Discussion and Concluding Remarks

Improved Color Algorithm for the $\mathcal{SSYN}\mathcal{C}$ (and $\mathcal{FSYN}\mathcal{C}$) Model. In the $\mathcal{SSYN}\mathcal{C}$ setting (and the $\mathcal{FSYN}\mathcal{C}$ setting), we need only two colors in Algorithm 1, which is optimal with respect to the number of colors when N is not known [12]. In particular, we do not need colors `Intermediate` and `Transit`. The colors `Intermediate` and `Transit` in Algorithm 1 are to synchronize the moves of the robots when there are two or more robots on a horizontal line $Hor(r_i)$ in the $\mathcal{ASYN}\mathcal{C}$ setting. However, in the $\mathcal{SSYN}\mathcal{C}$ (and also in the $\mathcal{FSYN}\mathcal{C}$) setting, this can be achieved without these colors since: (i) if only one robot r_i of $Hor(r_i)$ moves at round k , at round $k + 1$, the other robot r_j already sees r_i South of $Hor(r_i)$ and it can move in such a way that all the robots South of $Hor(r_i)$ see it; (ii) if both robots r_i, r_j on $Hor(r_i)$ move in round k , then at round $k + 1$ they will be on their final positions, all the robots South of $Hor(r_i)$ see both of them, and r_i, r_j see each other. All these results can be proved extending the analysis of Sect. 3 and runtime is still $\mathcal{O}(N)$.

Impact of Correctness of Lights after Faults. The tolerance to faults in our algorithm depends on the correctness of the colors of the lights even after robots experience (mobility) faults. I.e., even after robot becomes faulty, lights can be correctly set from `Off` to `Final`, possibly going through the changes to `Intermediate` and `Transit` (without moving). If the robot color stays as the color it has at the time of fault, then we cannot guarantee termination and also whether `COMPLETE VISIBILITY` is solved. This is because, it is difficult to determine for a robot r_i whether the (non-faulty) robots that are South of $Hor(r_i)$ already moved once or not. Therefore, it is an open problem to solve `COMPLETE VISIBILITY` in this setting tolerating multiple faults. The algorithm in [3] handles a single fault even when the light stays at the color at the time of fault.

Concluding Remarks. We have presented, to our best knowledge, the first fault-tolerant algorithm for the `COMPLETE VISIBILITY` problem using 4 colors for robots with lights in the $\mathcal{ASYN}\mathcal{C}$ setting under non-rigid movements and one-axis agreement, tolerating (up to) N faulty robots, not known a priori. The algorithm terminates in $\mathcal{O}(N)$ time avoiding collisions. The previous work [3] was only able to handle one faulty robot in the $\mathcal{SSYN}\mathcal{C}$ setting under rigid movements and both-axis agreement using 3 colors. We then showed that the number of colors can be improved from 4 to 2 in the $\mathcal{SSYN}\mathcal{C}$ setting (and also in the $\mathcal{FSYN}\mathcal{C}$) setting.

Many questions remain for future work. It will be interesting to minimize the number of colors from 4 to 2 in our algorithm in the *ASYNC* setting. Most importantly, it will be interesting to remove the one-axis agreement assumption and solve this problem tolerating multiple faults when lights can be faulty in addition to mobility faults.

References

1. Agathangelou, C., Georgiou, C., Mavronicolas, M.: A distributed algorithm for gathering many fat mobile robots in the plane. In: PODC, pp. 250–259 (2013)
2. Agmon, N., Peleg, D.: Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM J. Comput.* **36**(1), 56–82 (2006)
3. Aljohani, A., Sharma, G.: Complete visibility for mobile agents with lights tolerating a faulty agent. In: APDCM, pp. 834–843 (2017)
4. Bhagat, S., Chaudhuri, S.G., Mukhopadhyaya, K.: Fault-tolerant gathering of asynchronous oblivious mobile robots under one-axis agreement. *J. Discrete Algorith.* **36**, 50–62 (2016)
5. Bhagat, S., Chaudhuri, S.G., Mukhopadhyaya, K.: Formation of general position by asynchronous mobile robots under one-axis agreement. In: Kaykobad, M., Petreschi, R. (eds.) WALCOM 2016. LNCS, vol. 9627, pp. 80–91. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30139-6_7
6. Cohen, R., Peleg, D.: Local spreading algorithms for autonomous robot systems. *Theor. Comput. Sci.* **399**(1–2), 71–82 (2008)
7. Cord-Landwehr, A., Degener, B., Fischer, M., Hüllmann, M., Kempkes, B., Klaas, A., Kling, P., Kurras, S., Märtens, M., Meyer auf der Heide, F., Raupach, C., Swierkot, K., Warner, D., Weddemann, C., Wonisch, D.: A new approach for analyzing convergence algorithms for mobile robots. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6756, pp. 650–661. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22012-8_52
8. Czyzowicz, J., Gasieniec, L., Pelc, A.: Gathering few fat mobile robots in the plane. *Theor. Comput. Sci.* **410**(6–7), 481–499 (2009)
9. Das, S., Flocchini, P., Prencipe, G., Santoro, N., Yamashita, M.: Autonomous mobile robots with lights. *Theor. Comput. Sci.* **609**, 171–184 (2016)
10. D’Emidio, M., Frigioni, D., Navarra, A.: Characterizing the computational power of anonymous mobile robots. In: ICDCS, pp. 293–302 (2016)
11. Flocchini, P., Prencipe, G., Santoro, N.: Distributed computing by oblivious mobile robots. *Synth. Lectur. Distrib. Comput. Theor.* **3**(2), 1–185 (2012)
12. Luna, G.A.D., Flocchini, P., Chaudhuri, S.G., Poloni, F., Santoro, N., Viglietta, G.: Mutual visibility by luminous robots without collisions. *Inf. Comput.* **254**, 392–418 (2017)
13. Di Luna, G.A., Flocchini, P., Gan Chaudhuri, S., Santoro, N., Viglietta, G.: Robots with lights: overcoming obstructed visibility without colliding. In: Felber, P., Garg, V. (eds.) SSS 2014. LNCS, vol. 8756, pp. 150–164. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11764-5_11
14. Pagli, L., Prencipe, G., Viglietta, G.: Getting close without touching: Near-gathering for autonomous mobile robots. *Distrib. Comput.* **28**(5), 333–349 (2015)
15. Peleg, D.: Distributed coordination algorithms for mobile robot swarms: new directions and challenges. In: Pal, A., Kshemkalyani, A.D., Kumar, R., Gupta, A. (eds.) IWDC 2005. LNCS, vol. 3741, pp. 1–12. Springer, Heidelberg (2005). https://doi.org/10.1007/11603771_1

16. Sharma, G., Busch, C., Mukhopadhyay, S.: Mutual visibility with an optimal number of colors. In: Bose, P., Gaşieniec, L.A., Römer, K., Wattenhofer, R. (eds.) ALGOSENSORS 2015. LNCS, vol. 9536, pp. 196–210. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-28472-9_15
17. Sharma, G., Vaidyanathan, R., Trahan, J.L.: Constant-time complete visibility for asynchronous robots with lights. In: Spirakis, P., Tsigas, P. (eds.) SSS 2017. LNCS, vol. 10616, pp. 265–281. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69084-1_18
18. Sharma, G., Vaidyanathan, R., Trahan, J.L., Busch, C., Rai, S.: Complete visibility for robots with lights in $O(1)$ time. In: Bonakdarpour, B., Petit, F. (eds.) SSS 2016. LNCS, vol. 10083, pp. 327–345. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49259-9_26
19. Sharma, G., Vaidyanathan, R., Trahan, J.L., Busch, C., Rai, S.: Logarithmic-time complete visibility for asynchronous robots with lights. In: IPDPS, pp. 513–522 (2017)
20. Vaidyanathan, R., Busch, C., Trahan, J.L., Sharma, G., Rai, S.: Logarithmic-time complete visibility for robots with lights. In: IPDPS, pp. 375–384 (2015)

A Simple, Fast, Filter-Based Algorithm for Circular Sequence Comparison

Md. Aashikur Rahman Azim^(✉), Mohimenu Kabir, and M. Sohel Rahman

Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka 1215, Bangladesh
aashikazim@gmail.com, mahibuet045@gmail.com, sohel.kcl@gmail.com

Abstract. This paper deals with the circular sequence comparison problem, a fundamental step in many important tasks in bioinformatics, which appears as an interesting problem in many biological contexts. Traditional algorithms for measuring approximation in sequence comparison are based on the notions of distance or similarity, and are generally computed through sequence alignment techniques. The circular sequence comparison (CSC) problem consists in finding all comparisons of the rotations of a pattern \mathcal{P} of length m in a text \mathcal{T} of length n . In CSC, we consider comparisons with minimum distance from circular pattern \mathcal{P} to text \mathcal{T} under the *Hamming distance* model. In this paper, we present a simple and fast filter-based algorithm to solve the CSC problem. We compare our algorithm with the state of the art algorithms and the results are found to be excellent. In particular, our algorithm runs almost twice as fast than the state of the art. Much of the efficiency of our algorithm can be attributed to its filters that are effective but extremely simple and lightweight.

1 Introduction

Sequence alignment is a standard technique in bioinformatics for visualizing the relationships between residues in a collection of evolutionarily or structurally related proteins and DNA. In this paper, we consider the pairwise circular sequence comparison problem. Under the edit distance model, it consists in finding an optimal linear alignment of two circular strings. This problem for two strings \mathcal{P} and \mathcal{T} of length m and $n \geq m$, respectively, can be solved under the edit distance model.

The circular pattern, denoted $\mathcal{C}(\mathcal{P})$, corresponding to a given pattern $\mathcal{P} = \mathcal{P}_1 \dots \mathcal{P}_m$, is formed by connecting \mathcal{P}_1 with \mathcal{P}_m and forming a sort of a cycle; this gives us the notion where the same circular pattern can be seen as m different linear patterns, which would all be considered equivalent. In the sequence comparison (CSC) problem, the authors of [1] presented an algorithm based on the suffix array [2] to solve the CSC problem that finds the rotation of \mathcal{P} such that the β -blockwise q -gram distance between the rotated \mathcal{P} and \mathcal{T} is minimal, thereby solving exactly the circular sequence comparison problem under the β -blockwise q -gram distance measure. In this article, we are interested to solve the

problem CSC using the state of the art algorithm just stated above of [1] with a preprocessing of filtering techniques.

1.1 Applications and Motivations

Circular patterns are known to occur in the DNA of viruses [3,4], bacteria [5], eukaryotic cells [6], and archaea [7]. Gusfield [8] rightfully identified the algorithms on circular strings to be important in the analysis of organisms with such structures.

Double-stranded, circular chromosomes and plasmids are found in most bacteria and archaea. Whole-genome comparison is a very useful tool in classifying bacterial strains, as well as in inferring phylogenetic associations among them. This is due to the dense structure of bacterial chromosomes, caused by the absence of introns, and the organisation of genes into operons. The extended benefit of aligning plasmids is the ability to identify important genes, such as antibiotic resistance genes, thereby enabling their study and exploitation by genetic engineering techniques [9].

Circular strings have been studied in the context of sequence alignment. In [10], basic algorithms for pair wise and multiple circular sequence alignment have been presented. These results have later been improved in [11], where an additional preprocessing stage is added to speed up the execution time of the algorithm. Lee et al. [12] have considered Hamming distance and have presented efficient algorithms for finding the optimal alignment and consensus sequence of circular sequences on this distance metric. Interested readers are referred to [3–8,10–12] and references therein for further discussion on applications of circular pattern matching in computational molecular biology and other areas.

1.2 Our Contribution

In this paper, we present a fast and efficient algorithm for the exact Circular Sequence Comparison (CSC) problem based on some filtering techniques. In our prior work [13] we presented SimpLiFiCPM that is a filter-based algorithm to handle the exact version of the circular pattern matching (CPM) problem. SimpLiFiCPM is a filter-based algorithm that can solve the CPM problem very efficiently and can beat the state of the art algorithms by a good margin. However, we will introduce the same filtering techniques to preprocess the circular sequences and the DNA sequences. Our algorithm, referred to as the SFF-CSC algorithm (pronounced “Simple, Fast, Filter-based CSC Algorithm”) henceforth, employs a number of simple and effective filters to preprocess the given sequences. After this preprocessing, we get a text of reduced length on which we can apply any existing state of the art algorithms to find the solution of CSC problem. In summary, SFF-CSC algorithm, in some sense simplifies the search space of the exact circular sequence comparison problem.

We have conducted extensive experiments to compare our SFF-CSC algorithm with the state of the art algorithms and the results are found to be excellent. In particular, SFF-CSC algorithm runs almost twice as fast than the state

of the art. Our algorithm turns out to be much faster in practice because of the huge reduction in the search space through filtering. Also, the filtering techniques we use are simple and lightweight but as can be realized from the results, extremely effective.

1.3 Road Map

The rest of the paper is organized as follows. Section 2 gives a preliminary description of some terminologies and concepts related to stringology that will be used throughout this paper. Section 3 presents a brief literature review. In Sect. 4 we describe our filtering algorithms. Section 5 presents the experimental results. Section 6 draws conclusion followed by some future research directions.

2 Preliminaries

Let Σ be a finite *alphabet*. A string is a sequence of zero or more symbols from alphabet Σ . The set of all strings over Σ is denoted by Σ^* . The length of a string w is denoted by $|w|$. The empty string ϵ is a string of length 0, that is, $|\epsilon| = 0$. Let $\Sigma^+ = \Sigma^* - \{\epsilon\}$. For a string $w = xyz$, x , y and z are called a *prefix*, *factor* (or equivalently, *substring*), and *suffix* of w , respectively. The i -th character of a string w is denoted by $w[i]$ for $1 \leq i \leq |w|$, and the factor of a string w that begins at position i and ends at position j is denoted by $w[i : j]$ for $1 \leq i \leq j \leq |w|$. For convenience, we assume $w[i : j] = \epsilon$ if $j < i$. A k -factor is a factor of length k . A *suffix array* of string w of length $|w|$ is defined by an integer array of size $|w|$ which stores the starting positions of all lexicographically sorted suffixes of w . Again, let Σ^q denote the set of all strings of length q over Σ for $q = 1, 2, \dots, \infty$. A q -gram is any string $v = a_1a_2 \dots a_q$ in Σ^q .

A circular string of length m can be viewed as a traditional linear string which has the left-most and right-most symbols wrapped around and stuck together in some way. Under this notion, the same circular string can be seen as m different linear strings, which would all be considered equivalent. Given a string \mathcal{P} of length m , we denote by $\mathcal{P}^i = \mathcal{P}[i + 1 : m]\mathcal{P}[1 : i]$, $0 \leq i < m$, the i -th *rotation* of \mathcal{P} and $\mathcal{P}^0 = \mathcal{P}$.

Example 1. Suppose we have a pattern $\mathcal{P} = atcgatg$. The pattern \mathcal{P} has the following rotations (i.e., conjugates): $\mathcal{P}^1 = tcgatga$, $\mathcal{P}^2 = cgatgat$, $\mathcal{P}^3 = gatgatc$, $\mathcal{P}^4 = atgatcg$, $\mathcal{P}^5 = tgatcga$, $\mathcal{P}^6 = gatcgat$.

The *Hamming distance* between strings \mathcal{P} and \mathcal{T} , both of length n , is the number of positions i , $1 \leq i \leq n$, such that $\mathcal{P}[i] \neq \mathcal{T}[i]$. Given a non-negative integer k , we write $\mathcal{P} \equiv_k \mathcal{T}$ or equivalently say that \mathcal{P} k -matches \mathcal{T} , if the Hamming distance between \mathcal{P} and \mathcal{T} is at most k . In biology, the *Hamming distance* is sometimes referred to as the *Mutation distance*.

We consider the DNA alphabet, i.e., $\Sigma = \{a, c, g, t\}$. Following the approach of [13, 14], we assign a numeric value to each character of the alphabet as follows. Each character is assigned the number from the range $[1 \dots |\Sigma|]$ to the characters of

Σ following their inherent lexicographical order. We use $num(x), x \in \Sigma$ to denote the numeric value of the character x . So, we have $num(a) = 1, num(c) = 2, num(g) = 3$ and $num(t) = 4$. For a string S , we use the notation S_N to denote the numeric representation of the string S ; and $S_N[i]$ denotes the numeric value of the character $S[i]$. The concept of circular string and their rotations also apply naturally on their numeric representations as is illustrated in Example 2 below.

Example 2. Suppose we have a pattern $\mathcal{P} = atcgatg$. The numeric representation of \mathcal{P} is $\mathcal{P}_N = 1423143$. And this numeric representation has the following rotations: $\mathcal{P}_N^1 = 4231431, \mathcal{P}_N^2 = 2314314, \mathcal{P}_N^3 = 3143142, \mathcal{P}_N^4 = 1431423, \mathcal{P}_N^5 = 4314231, \mathcal{P}_N^6 = 3142314$.

We give some further definitions related to the problem we study [1, 15]. The q -gram profile of a string \mathcal{P} is the vector $\mathcal{G}_q(\mathcal{P})$, where $q > 0$ and $\mathcal{G}_q(\mathcal{P})[v]$ denotes the total number of occurrences of q -gram $v \in \Sigma^q$ in \mathcal{P} . The q -gram distance between two strings \mathcal{P} and \mathcal{T} is defined as

$$D_q(\mathcal{P}, \mathcal{T}) = \sum_{v \in \Sigma^q} |G_q(\mathcal{P})[v] - G_q(\mathcal{T})[v]|.$$

Note that D_q is a pseudo-metric as $D_q(\mathcal{P}, \mathcal{T})$ can be 0 even if $\mathcal{P} \neq \mathcal{T}$. And the β -blockwise q -gram distance two strings \mathcal{P} and \mathcal{T} of length m and n , respectively, is defined as

$$D_{\beta,q}(\mathcal{P}, \mathcal{T}) = \sum_{j=0}^{\beta-1} D_q(\mathcal{P}[\frac{j m}{\beta} \dots \frac{(j+1)m}{\beta} - 1], \mathcal{T}[\frac{j n}{\beta} \dots \frac{(j+1)n}{\beta} - 1]).$$

In this paper, we consider the following problem, where we search for the i -th rotation of \mathcal{P} that minimizes its blockwise distance from \mathcal{T} as defined in ([16]). Ties are broken arbitrarily.

Problem 1 [16] (Circular Sequence Comparison (CSC)). Given a pattern \mathcal{P} of length m , a text \mathcal{T} of length $n \geq m$, and integers $\beta \geq 1$ and $q < m$, find i such that $D_{\beta,q}(\mathcal{P}^i, \mathcal{T})$ is minimal.

The filters, we use related to the approach we study of [13] only give false positive [13].

3 Brief Literature Review

The blockwise circular sequence comparison problem for two strings \mathcal{P} and \mathcal{T} of length m and $n \geq m$, respectively, can be solved under the edit [17] distance model in time $\mathcal{O}(nm \log m)$ [18]. This blockwise distance framework is a really powerful filtering step for the actual circular edit distance computation [19]. Several other super-quadratic [20] and approximate quadratic-time [21] algorithms exist. Trivially, for molecular biology applications, the same problem can be solved in time $\mathcal{O}(nm^2)$, if the problem uses scoring matrices and affine gap

penalty scores. A direct application of pairwise circular sequence comparison is progressive multiple circular sequence alignment [10, 11, 22, 23]. Multiple circular sequence alignment has also been considered in [24] under the Hamming [25] distance model.

In [1], the authors introduced a fast exact algorithm for circular sequence comparison under some realistic model. We studied the β -blockwise q -gram distance between two strings \mathcal{P} and \mathcal{T} , that is, a more powerful generalization of the q -gram distance introduced as a string distance measure in [15]. Intuitively, and similarly to [17, 26], this generalization comprises partitioning \mathcal{P} and \mathcal{T} in β blocks each, as evenly as possible, computing the q -gram distance between the corresponding block pairs, and then summing up the distances computed blockwise. The authors of [1] presented an algorithm based on the suffix array [2] that finds the rotation of \mathcal{P} such that the β -blockwise q -gram distance between the rotated \mathcal{P} and \mathcal{T} is minimal, in time and space $\mathcal{O}(\beta m + n)$, where $m = |\mathcal{P}|$ and $n = |\mathcal{T}|$, thereby solving exactly the circular sequence comparison problem under the β -blockwise q -gram distance measure.

4 Filtering Algorithm

As has been mentioned above, our algorithm is based on some filtering techniques. In [13], the authors presented a framework based on a number of filters. Here, we will be using filters used by SimpLiFiCPM [13] to make it useful and effective in the context of CSC problem. In what follows, we follow the notations of [13]. We start with a brief overview of our approach below based on the filters of [13].

4.1 Overview of Our Approach

We employ a number of filters to compute a set \mathcal{N} of indexes of \mathcal{T} such that $\mathcal{C}(\mathcal{P})$ matches \mathcal{T} at position $i \in \mathcal{N}$ in such a way that there are no false negatives.

4.2 Filters of [13]

We employ a total of 6 filters of [13]. The key to observations of [13] and the resulting filters is the fact that each observation of the filters results in a unique output when applied to the rotations of a circular string. For example, consider a hypothetical function \mathcal{X} . We will always have the relation that $\mathcal{X}(\mathcal{P}) = \mathcal{X}(\mathcal{P}^i)$ for all $0 \leq i < n$. Recall that, \mathcal{P}^0 actually denotes \mathcal{P} . For the sake of conciseness, for such functions, we will abuse the notation a bit and use $\mathcal{X}(\mathcal{C}(\mathcal{P}))$ to represent $\mathcal{X}(\mathcal{P}^i)$ for all $0 \leq i < |\mathcal{P}|$.

Filter 1 of [13]. In [13], the authors defined the function $sum()$ on a string \mathcal{P} of length m as follows: $sum(\mathcal{P}) = \sum_{i=1}^m P_N[i]$. Filter 1, is based on this $sum()$ function which observation as follows.

Observation 1 [13]. Consider a circular string \mathcal{P} and a linear string \mathcal{T} both having length n . If $\mathcal{C}(\mathcal{P})$ matches \mathcal{T} , then we must have $sum(\mathcal{C}(\mathcal{P})) = sum(\mathcal{T})$.

Filters 2 and 3 of [13]. Following the notation of second and third filters of [13], the observation of filters 2 and 3 can be defined as follows. Filters 2 and 3, depend on a notion of distance between consecutive characters of a string. The *distance* between two consecutive characters of a string \mathcal{P} of length m is defined by $distance(\mathcal{P}[i], \mathcal{P}[i + 1]) = \mathcal{P}_N[i] - \mathcal{P}_N[i + 1]$, where $1 \leq i \leq m - 1$. The authors of [13] defined $total_distance(\mathcal{P}) = \sum_{i=1}^{m-1} distance(\mathcal{P}[i], \mathcal{P}[i + 1])$. The absolute version of same filter can be defined as: $abs_total_distance(\mathcal{P}) = \sum_{i=1}^{m-1} abs(distance(\mathcal{P}[i], \mathcal{P}[i + 1]))$, where $abs(x)$ returns the magnitude of x ignoring the sign. Before we apply these two functions on our strings to get our filters, we need to do a simple pre-processing on the respective string, i.e., \mathcal{P} in this case as follows. We extend the string \mathcal{P} by concatenating the first character of \mathcal{P} at its end. We use $ext(\mathcal{P})$ to denote the resultant string. So, we have $ext(\mathcal{P}) = \mathcal{P}\mathcal{P}[1]$. Since, $ext(\mathcal{P})$ can simply be treated as another string, we can easily extend the notation and concept of $\mathcal{C}(\mathcal{P})$ over $ext(\mathcal{P})$ and we continue to abuse the notation a bit for the sake of conciseness as mentioned at the beginning of Sect. 4.2 (just before Sect. 4.2).

The observations of Filter 2 and 3 of [13] are as follows.

Observation 2 [13]. Consider a circular string \mathcal{P} and a linear string \mathcal{T} both having length n and assume that $\mathcal{A} = ext(\mathcal{P})$ and $\mathcal{B} = ext(\mathcal{T})$. If $\mathcal{C}(\mathcal{P})$ matches \mathcal{T} , then, we must have $abs_total_distance(\mathcal{C}(\mathcal{A})) = abs_total_distance(\mathcal{B})$. Note carefully that the function $abs_total_distance()$ has been applied on the extended strings.

Observation 3 [13]. Consider a circular string \mathcal{P} and a linear string \mathcal{T} both having length n and assume that $\mathcal{A} = ext(\mathcal{P})$ and $\mathcal{B} = ext(\mathcal{T})$. If $\mathcal{C}(\mathcal{P})$ matches \mathcal{T} , then, we must have $total_distance(\mathcal{C}(\mathcal{A})) = total_distance(\mathcal{B})$. Note carefully that the function $total_distance()$ has been applied on the extended strings.

Filter 4 of [13]. Filter 4 uses the $sum()$ function used by Filter 1, albeit, in a slightly different way. In particular, it applies the $sum()$ function on individual characters. So, for $x \in \Sigma$ the authors of [13] defined $sum_x(\mathcal{P}) = \sum_{1 \leq i \leq |\mathcal{P}|, \mathcal{P}[i]=x} \mathcal{P}_N[i]$. The observation of filter 4 of [13] is as follows.

Observation 4 [13]. Consider a circular string \mathcal{P} and a linear string \mathcal{T} both having length n . If $\mathcal{C}(\mathcal{P})$ matches \mathcal{T} , then, we must have $sum_x(\mathcal{C}(\mathcal{P})) = sum_x(\mathcal{T})$ for all $x \in \Sigma$.

Filter 5 of [13]. Filter 5 depends on modulo operation between two consecutive characters. A modulo operation between two consecutive characters of a string \mathcal{P} of length m is defined as follows: $modulo(\mathcal{P}[i], \mathcal{P}[i + 1]) = \mathcal{P}_N[i] \% \mathcal{P}_N[i + 1]$, where $1 \leq i \leq m - 1$. The authors of [13] defined $sum_modulo(\mathcal{P})$ to be the summation of the results of the modulo operations on the consecutive characters of \mathcal{P} . More formally, $sum_modulo(\mathcal{P}) = \sum_{i=1}^{m-1} modulo(\mathcal{P}[i], \mathcal{P}[i + 1])$. Note that this observation is applied on the extended versions of the respective strings. The observation of filter 5 of [13] is as follows.

Observation 5 [13]. Consider a circular string \mathcal{P} and a linear string \mathcal{T} both having length n and assume that $\mathcal{A} = \text{ext}(\mathcal{P})$ and $\mathcal{B} = \text{ext}(\mathcal{T})$. If $\mathcal{C}(\mathcal{P})$ matches \mathcal{T} , then, we must have $\text{sum_modulo}(\mathcal{C}(\mathcal{A})) = \text{sum_modulo}(\mathcal{B})$. Note carefully that the function $\text{sum_modulo}()$ has been applied on the extended strings.

Filter 6 of [13]. In Filter 6 the author of [13] employed the $\text{xor}()$ operation. A bitwise exclusive-OR ($\text{xor}()$) operation between two consecutive characters of a string \mathcal{P} of length m is defined as follows: $\text{xor}(\mathcal{P}[i], \mathcal{P}[i + 1]) = \mathcal{P}_N[i] \wedge \mathcal{P}_N[i + 1]$, where $1 \leq i \leq m - 1$. They defined $\text{sum_xor}(\mathcal{P})$ to be the summation of the results of the xor operations on the consecutive characters of \mathcal{P} . More formally, $\text{sum_xor}(\mathcal{P}) = \sum_{i=1}^{m-1} \text{xor}(\mathcal{P}[i], \mathcal{P}[i + 1])$. Note that this observation is applied on the extended versions of the respective strings. The observation of filter 6 of [13] is as follows.

Observation 6 [13]. Consider a circular string \mathcal{P} and a linear string \mathcal{T} both having length n and assume that $\mathcal{A} = \text{ext}(\mathcal{P})$ and $\mathcal{B} = \text{ext}(\mathcal{T})$. If $\mathcal{C}(\mathcal{P})$ matches \mathcal{T} , then, we must have $\text{sum_xor}(\mathcal{C}(\mathcal{A})) = \text{sum_xor}(\mathcal{B})$. Note carefully that the function $\text{sum_xor}()$ has been applied on the extended strings.

4.3 The Approach of Our Algorithm

Now we present two algorithms to solve the exact CSC problem applying the six filters (Observations 1–6) presented above. It takes as input the pattern $\mathcal{P}[1 : m]$ of length m , the text $\mathcal{T}[1 : n]$ of length n , block-size β and q -gram size q . It calls Procedure *ECPS_FT* of [13] with $\mathcal{P}[1 : m]$ as the parameter and uses its output for exact CSC problem. (Procedure *ECPS_FT* is used to find the pattern signature of input pattern $\mathcal{P}[1 : m]$. This procedure takes circular pattern $\mathcal{P}[1 : m]$ as input and gives output the calculated values of all six observations [13].) We apply a sliding window approach with a window length of m and calculate the values applying the functions according to Observations 1 : 6 on the factor of \mathcal{T} captured by the window for exact CSC problem. Note that for Observations 2, 3, 5 and 6 we need to consider the extended string and hence the factor of \mathcal{T} within the window need be extended accordingly for calculating the values. After we calculate the values for a factor of \mathcal{T} , we check it against the returned values of Procedure *ECPS_FT*. If it matches, then we save the matched index and length bookkeeping purposes. Note that in case of overlapping factors (e.g., when the consecutive windows need to save), Procedure *SFF-CSC* saves only the non-overlapped characters.

Now note that we can compute the values of consecutive factors of \mathcal{T} using the sliding window approach quite efficiently as follows. For the first factor, i.e., $\mathcal{T}[1..m]$ we exactly follow the strategy of Procedure *ECPS_FT*. When it is done, we slide the window by one character and we only need to remove the contribution of the left most character of the previous window and add the contribution of the rightmost character of the new window. The functions are such that this can be done very easily using simple constant time operations. The only other

Algorithm 1. Algorithm SFF-CSC using procedure *ECPS_FT* of [13]

```

1: procedure SFF-CSC( $\mathcal{T}[1 : n]$ ,  $\mathcal{P}[1 : m], \beta, q$ )
2:   CALL ECPS_FT OF [13]( $\mathcal{P}[1 : m]$ )
3:   save the return value of observations 1 : 6 for further use here
4:   define an array of size 4 to keep fixed value of A, C, G, T
5:   initialize fixed array to {1, 2, 3, 4}
6:   lastIndex  $\leftarrow$  1
7:   startIndex  $\leftarrow$  -1
8:   length  $\leftarrow$  -1
9:   define vector pair bookShelf to save startIndex and length
10:  for  $i \leftarrow 1$  to  $m$  do
11:    calculate different filtering values in  $\mathcal{T}[1 : m]$  via observations 1 : 6 and make a
    running sum
12:  end for
13:  if 1 : 6 observations values of  $\mathcal{P}[1 : m]$  vs 1 : 6 observations values of  $\mathcal{T}[1 : m]$  have a
    match then
14:    ▷ Found a filtered match
15:    lastIndex  $\leftarrow$   $m$ 
16:    startIndex  $\leftarrow$  1
17:    length  $\leftarrow$   $m$ 
18:  end if
19:  for  $i \leftarrow 1$  to  $n - m$  do
20:    calculate different filtering values in  $\mathcal{T}[1 : m]$  via observations 1 : 6 by subtracting
     $i$ -th value along with wrapped value and adding  $i + m$ -th value and new wrapped value to
    the running sum
21:    if 1 : 6 filtering values of  $\mathcal{P}[1 : m]$  vs 1 : 6 filtering values of  $\mathcal{T}[i + 1 : i + m]$  have a
    match then
22:      ▷ Found a filtered match
23:      if  $i > \textit{lastIndex}$  then
24:        if  $\textit{startIndex} \neq -1$  then
25:          bookShelf.add(make_pair(startIndex, length))
26:        end if
27:        startIndex  $\leftarrow$   $i + 1$ 
28:        length  $\leftarrow$  0
29:      end if
30:      if  $i + m > \textit{lastIndex}$  then
31:        if  $i < \textit{lastIndex}$  then
32:           $j \leftarrow \textit{lastIndex} + 1$ 
33:        else
34:           $j \leftarrow i + 1$ 
35:        end if
36:        length  $\leftarrow$  length +  $(i + 1 + m) - j$ 
37:        lastIndex  $\leftarrow$   $i + m$ 
38:      end if
39:    end if
40:  end for
41: bookShelf.add(make_pair(startIndex, length))
42: minDistance  $\leftarrow$   $\infty$ 
43: saveI  $\leftarrow$  -1
44: for  $k \leftarrow 1$  to bookShelf.size() do
45:   CALL SACSC [1](bookShelf.get(pair( $k$ )),  $\beta, q$ )
46:   save the minimum distance in minDistance and corresponding index  $i$  in saveI
47: end for
48: Output saveI and minDistance
49: end procedure

```

issue that needs be taken care of is due to the use of the extended string in two of the filters. But this too does not need more than simple constant time operations. Therefore, overall runtime of the algorithm is $\mathcal{O}(m) + \mathcal{O}(n - m) = \mathcal{O}(n)$ up to line 43 in Procedure SFF-CSC (Algorithm 1). For rest of the Algorithm from line 44, we find reduced text \mathcal{T}' (say) after filtering where the start index and length of each reduced text are saved for bookkeeping purposes in Procedure SFF-CSC. At this point we can use any algorithm that can solve the CSC problem and apply it over \mathcal{T}' and output the best comparison which minimizes the q -gram distance based on the distance matrix described for Problem 1. Now, suppose we use Algorithm \mathcal{A} at this stage which runs in $\mathcal{O}(f(|\mathcal{T}'|))$ time. Then, clearly, the overall running time of our approach is $\mathcal{O}(n) + \mathcal{O}(f(|\mathcal{T}'|))$. In our implementation we have used *saCSC* the recent algorithm of [1] to solve the exact CSC problem. In particular, in [1], the authors presented a circular sequence comparison algorithm using suffix-array construction. They built a library to solve *CSC* problem. The library is freely available and can be found here: [27]. We only apply *saCSC* on the reduced string to solve exact version of the problem.

5 Experimental Results

We have implemented SFF-CSC and conducted extensive experiments to analyze its performance. We show the comparison based on the experimental result between saCSC of [1] and our algorithm. Algorithm saCSC [1] has been implemented as library functions in the *C* programming language under *GNU/Linux* operating system. The library implementation is distributed under the GNU General Public License (GPL). It takes as input the pattern \mathcal{P} of length m , the text \mathcal{T} of length n , and integers block-size, $\beta > 1$ and q -grams, $q < m$, and returns the rotation of \mathcal{P} for which blockwise q -gram distance is minimal with \mathcal{T} according to the Problem 1.

5.1 Datasets

We have used real genome data in our experiments as the text string, \mathcal{T} . This data has been collected from [28]. Here, we have taken 700 MB of data for our experiments. We have generated random patterns of different length by a random indexing technique in this 700 MB of text string.

5.2 Environment

We have conducted our experiments on a Samsung Laptop of Intel Core(TM) i5-2430M CPU @2.40 GHz processor product family and 4 GB of RAM under GNU/Linux. We have coded the SFF-CSC algorithm in C++ using a GNU compiler with General Public License (GPL). As has been mentioned already above, our implementation of the SFF-CSC algorithm uses the saCSC [1]. With the help of the library used in [1], we have compared the running time of saCSC of [1] and the SFF-CSC algorithm.

5.3 Experimental Results

Here we represent the main experimental results and comparisons between our algorithm SFF-CSC and saCSC of [1]. Table 1 reports the elapsed time and speed-up comparisons for various pattern sizes ($2000 \leq m \leq 1200000$) and for various q -grams sizes ($5 \leq q \leq 40$) and block-size $\beta = \sqrt{m}$. In [1], the authors presented that their algorithm saCSC performs better for the block size, $\beta = \sqrt{m}$. This is the reason why we set $\beta = \sqrt{m}$ in our experimental setup. As can be seen from Table 1, our algorithm runs faster than saCSC in all cases.

In order to analyze and understand the effect of our filters we have run a second set of experiments as follows. We have run experiments on three variants of SFF-CSC algorithm where the first variant (SFF-CSC-[1..3]) only employs Filters 1 through 3, the second variant (SFF-CSC-[1..4]) only employs Filters 1 through 4, and finally the third variant (SFF-CSC-[1..5]) employs Filters 1 through 5. This experiment result is available at online¹. From the result it can be checked that saCSC is able to beat SFF-CSC-[1..3] in a number of cases. However, SFF-CSC-[1..4] and SFF-CSC-[1..5] run significantly faster than saCSC

Table 1. Elapsed-time (in seconds) and speed-up comparison between saCSC [1] and our algorithm (exact CSC) considering all the six filters for a text of size 700 MB. Here, $\beta = \sqrt{m}$.

m	q	Elapsed time (s) of saCSC	Elapsed time (s) of SFF-CSC	Speed up of SFF-CSC	m	q	Elapsed time (s) of saCSC	Elapsed time (s) of SFF-CSC	Speed up of SFF-CSC
2000	5	41	22	1.9	250000	25	129	66	2
3000	5	42	21	2	300000	25	140	69	2
4000	5	45	25	1.8	350000	25	147	74	2
5000	5	44	23	1.9	400000	25	160	78	2.1
6000	5	47	25	1.9	450000	25	172	81	2.1
2000	10	39	18	2.2	500000	30	203	99	2.1
3000	10	44	20	2.2	550000	30	255	105	2.4
4000	10	47	26	1.8	650000	30	266	123	2.2
5000	10	52	28	1.9	700000	30	301	143	2.1
6000	10	57	26	2.2	750000	30	333	192	1.7
5000	15	60	31	1.9	800000	35	402	195	2.1
6000	15	63	32	2	850000	35	498	240	2.1
7000	15	60	29	2.1	900000	35	578	256	2.3
8000	15	66	32	2.1	950000	35	710	280	2.5
9000	15	67	34	2	1000000	35	880	423	2.1
10000	20	56	27	2.1	1100000	40	902	440	2.1
50000	20	63	33	1.9	1150000	40	935	510	1.8
100000	20	111	52	2.1	1200000	40	1020	550	1.9
150000	20	106	51	2.1	1200000	40	1022	530	1.9
200000	20	123	62	2	1200000	40	1045	501	2.1

¹ <https://goo.gl/bKZ52e>.

of [1] in all cases. This indicates that as more and more effective filters are imposed, our algorithm performs better.

6 Conclusions

In this paper, we have employed some effective lightweight filtering techniques to reduce the search space of the Circular Sequence Comparison (CSC) problem. We have presented SFF-CSC algorithm, an extremely fast algorithm based on the above-mentioned filters. Much of the speed of our algorithm comes from the fact that our filters are effective but extremely simple and lightweight. In our experiments, SFF-CSC has achieved a minimum of two-fold speed-up than the state of the art algorithms. The most intriguing feature of the SFF-CSC algorithm is perhaps its capability to plug in any algorithm to solve CSC and take advantage of it. We are now working towards adapting the filters so that it could work for the approximate (heuristic) version of CSC of [1].

References

1. Grossi, R., Iliopoulos, C.S., Mercas, R., Pisanti, N., Pissis, S.P., Retha, A., Vayani, F.: Circular sequence comparison: algorithms and applications. *Algorithms Mol. Biol.* **11**(1), 12 (2016)
2. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. *SIAM J. Comput.* **22**(5), 935–948 (1993)
3. Dulbecco, R., Vogt, M.: Evidence for a ring structure of polyoma virus DNA. *Proc. Natl. Acad. Sci.* **50**(2), 236–243 (1963)
4. Weil, R., Vinograd, J.: The cyclic helix and cyclic coil forms of polyoma viral DNA. *Proc. Natl. Acad. Sci.* **50**(4), 730–738 (1963)
5. Thanbichler, M., Wang, S., Shapiro, L.: The bacterial nucleoid: A highly organized and dynamic structure. *J. Cell Biochem.* **96**(3), 506–521 (2005)
6. Lipps, G.: *Plasmids: Current Research and Future Trends*. Caister Academic Press, Norfolk (2008)
7. Allers, T., Mevarech, M.: Archaeal genetics - the third way. *Nat. Rev. Genet.* **6**, 58–73 (2005)
8. Gusfield, D.: *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, New York (1997)
9. Del Castillo, C.S., Hikima, J., Jang, H.B., Nho, S.W., Jung, T.S., Wongtavatchai, J., Kondo, H., Hirono, I., Takeyama, H., Aoki, T.: Comparative sequence analysis of a multidrug-resistant plasmid from aeromonas hydrophila. *Antimicrob. Agents Chemother.* **57**(1), 120–129 (2013)
10. Mosig, A., Hofacker, I., Stadler, P., Zell, A.: Comparative analysis of cyclic sequences: viroids and other small circular RNAs. *German Conference on Bioinformatics. LNI*, vol. 83, pp. 93–102 (2006)
11. Fernandes, F., Pereira, L., Freitas, A.: CSA: an efficient algorithm to improve circular DNA multiple alignment. *BMC Bioinform.* **10**, 1–13 (2009)
12. Lee, T., Na, J.C., Park, H., Park, K., Sim, J.S.: Finding optimal alignment and consensus of circular strings. In: Amir, A., Parida, L. (eds.) *CPM 2010. LNCS*, vol. 6129, pp. 310–322. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13509-5_28

13. Azim, M.A.R., Iliopoulos, C.S., Rahman, M.S., Samiruzzaman, M.: SimPLiFiCPM: a simple and lightweight filter-based algorithm for circular pattern matching. *Int. J. Genomics* **2015**, 10 (2015). Article ID 259320
14. Azim, M.A.R., Iliopoulos, C.S., Rahman, M.S., Samiruzzaman, M.: A fast and lightweight filter-based algorithm for circular pattern matching. In: *ACM Conference on Bioinformatics, Computational Biology, and Health Informatics* (2014)
15. Ukkonen, E.: Approximate string-matching with q-grams and maximal matches. *Theor. Comput. Sci.* **92**(1), 191–211 (1992)
16. Helinski, D.R., Clewell, D.: Circular DNA. *Annu. Rev. Biochem.* **40**(1), 899–942 (1971)
17. Peterlongo, P., Sacomoto, G.A.T., do Lago, A.P., Pisanti, N., Sagot, M.-F.: Lossless filter for multiple repeats with bounded edit distance. *Algorithms Mol. Biol.* **4**(1), 3 (2009)
18. Maes, M.: On a cyclic string-to-string correction problem. *Inf. Process. Lett.* **35**(2), 73–78 (1990)
19. Ayad, L.A., Barton, C., Pissis, S.P.: A faster and more accurate heuristic for cyclic edit distance computation. *Pattern Recogn. Lett.* **88**(Suppl. C), 81–87 (2017)
20. Marzal, A., Barrachina, S.: Speeding up the computation of the edit distance for cyclic strings. In: *Proceedings of the 15th International Conference on Pattern Recognition*, vol. 2, pp. 891–894 (2000)
21. Bunke, H., Bhlér, U.: Applications of approximate string matching to 2D shape recognition. *Pattern Recogn.* **26**(12), 1797–1812 (1993)
22. Barton, C., Iliopoulos, C.S., Kundu, R., Pissis, S.P., Retha, A., Vayani, F.: Accurate and efficient methods to improve multiple circular sequence alignment. In: Bampis, E. (ed.) *SEA 2015. LNCS*, vol. 9125, pp. 247–258. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20086-6_19
23. Ayad, L.A.K., Pissis, S.P.: MARS: improving multiple circular sequence alignment using refined sequences. *BMC Genomics* **18**(1), 86 (2017)
24. Lee, T., Na, J.C., Park, H., Park, K., Sim, J.S.: Finding consensus and optimal alignment of circular strings. *Theor. Comput. Sci.* **468**, 92–101 (2013)
25. Peterlongo, P., Pisanti, N., Boyer, F., do Lago, A.P., Sagot, M.F.: Lossless filter for multiple repetitions with hamming distance. *J. Discrete Algorithms* **6**(3), 497–509 (2008)
26. Rasmussen, K.R., Stoye, J., Myers, E.W.: Efficient q-gram filters for finding all ε -matches over a given length. *J. Comput. Biol.* **13**(2), 296–308 (2006)
27. <https://github.com/solonas13/csc>
28. <http://hgdownload-test.cse.ucsc.edu/goldenPath/hg19/bigZips/>

Boosting over Non-deterministic ZDDs

Takahiro Fujita¹(✉), Kohei Hatano^{2,3}, and Eiji Takimoto¹

¹ Department of Informatics, Kyushu University, Fukuoka, Japan
{[takahiro.fujita](mailto:takahiro.fujita@inf.kyushu-u.ac.jp), [eiji](mailto:eiji@inf.kyushu-u.ac.jp)}@inf.kyushu-u.ac.jp

² Faculty of Arts and Science, Kyushu University, Fukuoka, Japan
hatano@inf.kyushu-u.ac.jp

³ RIKEN AIP, Tokyo, Japan

Abstract. We propose a new approach to large-scale machine learning, *learning over compressed data*: First compress the training data somehow and then employ various machine learning algorithms on the compressed data, with the hope that the computation time is significantly reduced when the training data is well compressed. As the first step, we consider a variant of the Zero-Suppressed Binary Decision Diagram (ZDD) as the data structure for representing the training data, which is a generalization of the ZDD by incorporating non-determinism. For the learning algorithm to be employed, we consider boosting algorithm called AdaBoost* and its precursor AdaBoost. In this work, we give efficient implementations of the boosting algorithms whose running times (per iteration) are linear in the size of the given ZDD.

1 Introduction

Most tasks in machine learning are formulated as optimization problems of various types. Recently, the amount of data to be treated is growing enormously large, and so the demands on scalable optimization methods are increasing. Probabilistic approach such as stochastic gradient descent methods [3] is now widely employed as standard techniques for large scale machine learning. Obviously, these methods require the time and/or the space complexity to be proportional to the size of given data.

In this paper, we propose a new approach: *learning over compressed data*. That is, we first compress the given data somehow, and then employ various machine learning algorithms on the compressed data without explicitly reconstructing the original data. To be more precise, for any target machine learning algorithm to be employed, we apply an efficient algorithm running over the compressed data, which simulates the behavior of the target algorithm running over the original data, with the hope that the time and space complexity are significantly reduced when the data is well compressed. Although the complexity for compressing data of the first phase needs to be sufficiently small, we can expect great improvement of time and space complexity, especially when high compression ratio is achieved.

The methodology of working over compressed data has gained much attention in the areas of database and data mining, where various methods have been developed, say, for the string search from a compressed string and the frequent word extraction from compressed texts [2, 4, 8]. But, as far as the authors are aware, most of all the methods developed so far are limited to simple tasks such as search and counting, and few results are known for more complex tasks such as optimization in machine learning. Notable exceptions contain the results of Nishino et al. [14] and Tabei et al. [18], respectively. Their methods use string compression techniques to perform matrix-based computations under small memory spaces. Our method, we will show later, is completely different from theirs.

As the first step toward establishing a general methodology of learning from compressed data, we consider a variant of the Zero-Suppressed Binary Decision Diagram (ZDD) as the data structure for representing the training data. The ZDD is a general data structure for representing a family of sets [7, 10], and is appropriate for our purpose. One of the reasons is that many results are reported in the literature that the ZDD indeed has ability of compactly representing various data in various domains [5, 11, 12].

In this paper, we slightly generalize the ZDD by incorporating non-determinism and propose a new data structure called the non-deterministic ZDD (NZDD, for short). The NZDD has more flexibility for representing data because of the non-determinism. Also, our efficient simulation algorithms (showed later) fit naturally to the NZDDs. On the other hand, it seems to be NP-hard to construct an NZDD of minimal size from a given training data. An efficient construction method of succinct NZDDs is left as future work.

For the learning algorithm to be employed over the NZDD representation of the training data, we consider a boosting algorithm called the AdaBoost* [16]. The AdaBoost* is a refined version of the seminal boosting algorithm AdaBoost [1] and is guaranteed to find a hyperplane that maximizes the margin. In this paper, we give an efficient implementation of the AdaBoost*. Its running time (per iteration) does not depend on the size of training data but is only linear in the size of the given NZDD. In addition, our proposed framework can be applicable to the AdaBoost as well and a similar guarantee also holds.

So, our method takes advantage when the size of NZDD is much smaller than the size of the training data, provided that the time complexity of constructing the NZDD is moderately small.

2 Problem Statement and AdaBoost*

First we describe the problem of 1-norm hard margin maximization and then briefly review the AdaBoost* which is one of the boosting algorithms that solve the problem.

2.1 1-Norm Hard Margin Maximization

Let X be a set called the instance space, and assume that we are given a finite set of *base hypotheses* $H = \{h_1, h_2, \dots, h_n\} \subseteq \{h : X \rightarrow \{0, 1\}\}$. Note that the base hypotheses are usually assumed to take values in $\{-1, 1\}$, but since any function $g : X \rightarrow \{-1, 1\}$ can be represented as the difference of 0–1 valued functions (e.g., $g(x) = \mathbf{1}[g(x) = 1] - \mathbf{1}[g(x) = -1]$), we can assume 0–1 valued hypotheses without loss of generality. The base hypothesis class H defines a feature map, which maps any instance $x \in X$ to the feature vector $(h_1(x), h_2(x), \dots, h_n(x))$ in the feature space $\{0, 1\}^n$. Later we will regard the feature vector for x as the set $H(x) = \{h_j \mid h_j(x) = 1\}$ and thanks to the assumption above, any base hypothesis $h_j \notin H(x)$ takes value 0 for x , which is a crucial property that makes our algorithm work.

Now we give the problem statement of 1-norm hard margin maximization. The input is a *sample* $S = \{(x_1, y_1), \dots, (x_m, y_m)\} \subseteq X \times \{-1, 1\}$, where x_i for $y_i = 1$ is called a positive instance and x_i for $y_i = -1$ a negative instance, and the output is a hyperplane in the feature space that separates the positive instances from the negative instances as much as possible. More precisely, the goal is to find

$$\alpha^* = \arg \max_{\alpha \in \{\mathbb{R}^n \mid \|\alpha\|_1 = 1\}} \min_{1 \leq i \leq m} y_i \sum_{j=1}^n \alpha_j h_j(x_i). \quad (1)$$

We denote by $\alpha \in \{\mathbb{R}^n \mid \|\alpha\|_1 = 1\}$ the hyperplane whose normal vector is α , which also represents the convex combination of base hypotheses $f(x) = \sum_{j=1}^n \alpha_j h_j(x)$. Note that since the 1-norm of α is normalized, $|f(x)|$ denotes the distance of the feature vector $(h_1(x), \dots, h_n(x))$ to the hyperplane α measured by ∞ -norm. Thus, the signed distance $y_i f(x_i)$ (which is positive if and only if f correctly classifies x_i) is called the margin of the hyperplane α with respect to the labeled instance (x_i, y_i) . Let $\rho = \min_i y_i f(x_i)$ be the minimum margin of α over all labeled instances in the sample. Note that α^* is the hyperplane that maximizes ρ . It is well known that if $\rho > 0$, which means that the sample S is linearly separable, then the combined hypothesis f has a generalization error bound that is proportional to $1/\rho$ [9]. So, the goal of maximizing ρ is natural. Let $\rho^* = \min_i y_i \sum_j \alpha_j^* h_j(x_i)$ be the optimal margin.

In what follows, we assume without loss of generality that all labeled feature vectors $(h_1(x_i), \dots, h_n(x_i), y_i)$ are distinct.

2.2 AdaBoost*

The optimization problem (1) can be formulated as a linear programming problem of size $O(nm)$ and hence efficiently solved by an LP solver. However, in many cases, the number n of base hypotheses is very large (sometimes infinite), and thus the problem is infeasible for LP solvers. In such cases, boosting may provide an alternative way. In particular, the AdaBoost* of Rätsch and Warmuth [16] provably converges to the maximum margin ρ^* within precision ν in $2 \log(\frac{m}{\nu^2})$

Algorithm 1. AdaBoost*

Input $S = \{(x_1, y_1), \dots, (x_m, y_m)\} \subseteq X \times \{-1, 1\}$

Output f

1. Let $\alpha_j = 0$ for $j = 1, \dots, n$
 2. Let $d_1(i) = 1/m$ for $i = 1, \dots, m$
 3. For $t = 1, \dots, T$
 - (a) Compute the edges
 $\gamma_{t,j} = \sum_{i=1}^m d_t(i) y_i h_j(x_i)$ for $j = 1, \dots, n$.
 - (b) Let $j_t = \arg \max_{1 \leq j \leq n} |\gamma_{t,j}|$; $\gamma_t = \gamma_{t,j_t}$.
 - (c) Set $\rho_t = \min_{r=1, \dots, t} |\gamma_r| - \nu$;
 - (d) Update coefficients $\alpha_{j_t} = \alpha_{j_t} + \frac{1}{2} \log \frac{1+\gamma_t}{1-\gamma_t} - \frac{1}{2} \log \frac{1+\rho_t}{1-\rho_t}$
 - (e) Update weights
 $d_{t+1}(i) = d_t(i) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)) / Z_t$
for $i = 1, \dots, m$, where
 $Z_t = \sum_{i=1}^m d_t(i) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i))$
 4. Let $f(x) = \sum_{j=1}^n \frac{\alpha_j}{\|\alpha\|_1} h_j(x)$
-

iterations. Below we describe how the AdaBoost* behaves when applied to the base hypothesis class H . On each round $t = 1, 2, \dots, T$, it (i) computes a distribution d_t over the sample S , (ii) finds a base hypothesis $h_{j_t} \in H$ with the maximum *edge* (average margin) with respect to d_t , and (iii) updates the coefficient α_{j_t} . Finally, normalizing the coefficient α , it obtains a final hypothesis f . A pseudocode is given in Algorithm 1, where part (ii) above is implemented in a very naive manner: compute the edges of all base hypotheses (line 3-(a)) and then choose the maximum among them (line 3-(b)). So, this implementation is inefficient for a very large n . But, AdaBoost* (and any other boosting algorithm) has a considerable advantage over LP solvers when the hypothesis class H has an efficient implementation, called the base learner, for this part: to find a base hypothesis with the maximum edge from a given distribution over the sample. In this case, the two lines (3-(a) and 3-(b)) are replaced by the base learner. The next theorem shows a performance guarantee of the AdaBoost*.

Theorem 1 (Rätsch and Warmuth [16]). *If $T \geq \frac{2 \log m}{\nu^2}$, then AdaBoost* (Algorithm 1) outputs a combined hypothesis f such that $\min_{1 \leq i \leq m} y_i f(x_i) \geq \rho^* - \nu$.*

In this paper, we consider the situation where the size n of H is small but the sample size m is very large, as is often the case, and both the direct applications of LP solvers and the AdaBoost* may be useless.

2.3 AdaBoost

The AdaBoost, proposed by Freund and Schapire [1], is a precursor of the AdaBoost*. The algorithm, unlike the AdaBoost*, is not shown to provably maximize the hard margin. However, it is shown that it achieves at least half of the

maximum hard margin asymptotically under weak technical conditions [15, 16]. Besides, the AdaBoost is much more popular because of its simplicity and the empirical performances. The behavior of the AdaBoost is almost the same as the AdaBoost*. More precisely, instead of 3. (c) and (d) in Algorithm 1, the AdaBoost updates the coefficient as $\alpha_{j_t} = \alpha_{j_t} + \frac{1}{2} \log \frac{1+\gamma_t}{1-\gamma_t}$. Therefore, the theoretical results we will show also are applicable to the AdaBoost.

3 A Dag Representation for Samples

As a data structure for storing the sample, we propose a dag representation for a family of sets called the non-deterministic ZDD (NZDD, for short). It can be seen as a generalization of the ZDD by incorporating non-determinism.

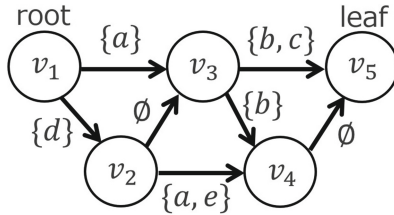


Fig. 1. An NZDD representation for $\{\{a, b\}, \{a, b, c\}, \{a, d, e\}, \{b, c, d\}, \{b, d\}\}$

3.1 Non-deterministic ZDD (NZDD)

An NZDD is specified by a 4-tuple $G = (V, E, \Sigma, \Phi)$, where (V, E) is a directed acyclic graph with a single root and a single leaf, Σ is a ground set, and $\Phi : E \rightarrow 2^\Sigma$ is a function that assigns to each edge e a subset $\Phi(e)$ of Σ . Note that $\Phi(e)$ can be the empty set \emptyset . Furthermore we require the additional properties as described below. Let \mathcal{P}_G be the set of all paths from the root to the leaf in G , where a path P in \mathcal{P}_G is specified by the set of edges in P , i.e., $P \subseteq E$.

1. Every path $P \in \mathcal{P}_G$ represents a subset $S(P) \subseteq \Sigma$ defined as $S(P) = \bigcup_{e \in P} \Phi(e)$. Thus, the NZDD G defines a subset family as $L(G) = \{S(P) \mid P \in \mathcal{P}_G\} \subseteq 2^\Sigma$.
2. For every pair of paths $P, P' \in \mathcal{P}_G$, $S(P) \neq S(P')$ if $P \neq P'$.
3. For every path $P \in \mathcal{P}_G$, $\Phi(e) \cap \Phi(e') = \emptyset$ for any $e, e' \in P$ with $e \neq e'$.

Note that by the second property, there exists a one-to-one correspondence between the set of paths \mathcal{P}_G and the subset family $L(G)$. In particular, we have $|\mathcal{P}_G| = |L(G)|$. The third property says that every element $a \in \Sigma$ appears at most once in every path $P \in \mathcal{P}_G$. That is, letting $E(a) = \{e \in E \mid a \in \Phi(e)\}$, we have $|E(a) \cap P| \leq 1$ for every $P \in \mathcal{P}_G$. Finally, we define the size of G as $|G| = \sum_{e \in E} |\Phi(e)|$. Note that the size $|G|$ can be significantly small as compared with the number of paths $|\mathcal{P}_G|$. In other words, the NZDD G is a compact representation for the subset family $L(G)$. As an example, we give in Fig. 1 an NZDD that represents a subset family.

3.2 NZDD Representation for the Sample

Now we describe how we represent the sample S as an NZDD.

Recall that $H(x) = \{h_j \in H \mid h_j(x) = 1\}$ for each instance $x \in X$. Let $Z^+ = \{H(x_i) \mid (x_i, 1) \in S\}$ and $Z^- = \{H(x_i) \mid (x_i, -1) \in S\}$ be the subset families with the ground set $\Sigma = H$, which correspond to the positive and the negative instances in the sample S , respectively. Let G^+ and G^- be NZDDs for the families Z^+ and Z^- , respectively. That is, $L(G^+) = Z^+$ and $L(G^-) = Z^-$. Finally, the NZDD G for the sample S is obtained by (i) putting an additional node as the global root with two outgoing edges labeled with \emptyset , where one edge is connected to the root of G^+ and the other is to the root of G^- , and (ii) merging the leaves of G^+ and G^- to a single leaf (See Fig. 2 for example). Note that G is not necessarily a minimal NZDD even if G^+ and G^- are minimal, because G may be further simplified by merging a node in G^+ and a node in G^- . But, we define G in this way, so that any path in G^+ and any path in G^- are disjoint.

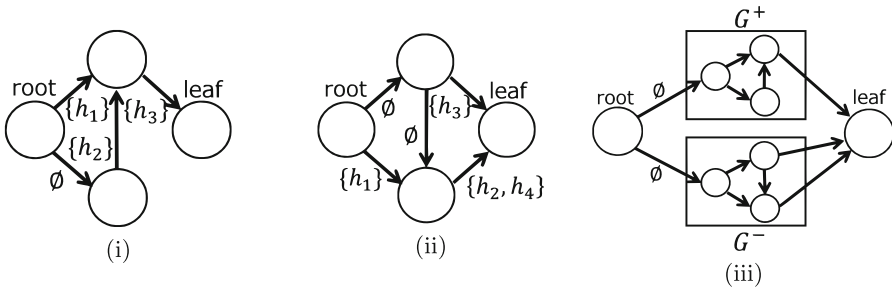


Fig. 2. (i) An NZDD G^+ for $Z^+ = \{\{h_1, h_3\}, \{h_2, h_3\}\}$; (ii) An NZDD G^- for $Z^- = \{\{h_1, h_2, h_4\}, \{h_2, h_4\}, \{h_3\}\}$; (iii) An NZDD for the sample consisting of positive instances Z^+ and negative instances Z^-

3.3 Relations to ZDDs and NFAs

We show that the ZDD representation is a special case of the NZDD representation. To see this, we consider the class of NZDDs of the following form:

1. Each edge e is labeled with either a singleton or the empty set. That is, $|\Phi(e)| \leq 1$.
2. Each internal node has one or two outgoing edges. If it has two outgoing edges, one of them is labeled with the empty set.
3. There exists a fixed ordering over Σ such that for any pair of edges e and e' labeled with singletons $\{a\}$ and $\{a'\}$, respectively, if e is an ancestor of e' , then a precedes a' in this ordering.

It is easy to see that any ZDD can be seen as an NZDD in this form.

Conversely, consider the class of NZDDs of the following form:

1. It is ordered. That is, the third condition above is satisfied.
2. For each pair of edges e and e' outgoing from a common node, $\Phi(e) \cap \Phi(e') = \emptyset$.

Then, we can show that any NZDD of this class has an equivalent ZDD of the same size. So, only the difference of ordered NZDDs from ZDDs is that we allow non-determinism, i.e., $\Phi(e) \cap \Phi(e') \neq \emptyset$.

Next we consider the relation of ordered NZDDs to NFAs. Under the ordering over Σ , we can identify a subset $\{a_1, a_2, \dots, a_k\} \subseteq \Sigma$ with a string $a_1 a_2 \dots a_k \in \Sigma^*$ over the alphabet Σ , where $a_1 < a_2 < \dots < a_k$ under the ordering $<$. Note that the empty set corresponds to the empty string ϵ . In this way, a subset family can be seen as a language. From this viewpoint, we can regard an NZDD G as an NFA that recognizes the language $L(G)$, with the root identified with the start state and the leaf with the unique accepting state. The difference is that, in the NZDD representation, we have only a single accepting path for each string in the language. This implies that any DFA for such a language can be converted to an NZDD in an obvious way. Note that in order to make the accepting state unique, we may need to put an additional leaf and connecting every accepting state to the leaf by an additional edge labeled with the empty set (ϵ -transition).

3.4 Complexity of Constructing NZDDs

When given a subset family $L \subseteq 2^\Sigma$, we want to compute a minimal NZDD G with $L(G) = L$. So far, the time complexity of the problem is unknown, but it seems to be NP-hard because so are the closely related problems, namely, construction of a minimal ZDD (over all ordering) [7] and construction of a minimal NFA [6]. On the other hand, we have a polynomial time algorithm for constructing a minimal ZDD when given an ordering [17] and a linear time algorithm for constructing a minimal DFA for a finite language [17]. So, practically, we can use these algorithms for constructing an ordered NZDD of small size.

4 Simulating AdaBoost* over an NZDD Representation for The sample

In this section, we give an algorithm that efficiently simulates the AdaBoost* over an NZDD G that represents a sample $S = \{(x_1, y_1), \dots, (x_m, y_m)\}$, without explicitly reconstructing the sample S from G . In particular, the running time (per iteration) of our algorithm does not depend on the sample size m but is linear in the size of G . First we state the main theorem.

Theorem 2. *There exists an algorithm that, when given an NZDD G that represents a sample S , exactly simulates AdaBoost* whose running time is $O(|G|)$ per iteration.*

So, if the sample is significantly compressed in the NZDD representation, our algorithm runs much faster than the direct application of the AdaBoost* when the computation time of constructing G from S is negligible. More specifically, if we use a linear time algorithm for constructing an NZDD from a minimal DFA as described in the previous section, then the total running time of our algorithm is $O(nm + T|G|)$, whereas the total running time of the direct application of the AdaBoost* is $O(nmT)$. So, if $|G| \ll nm$, then our algorithm would be faster¹.

Further, since the AdaBoost is almost identical to the AdaBoost* in an algorithmic sense, we have the following corollary as well.

Corollary 3. *There exists an algorithm that, when given an NZDD G representing S , simulates AdaBoost whose running time is $O(|G|)$ per iteration.*

Below we describe a basic idea of the algorithm. Obviously, we cannot explicitly maintain the distribution d_t over the sample S . Instead, we maintain one weight $w_{t,e}$ for each edge e of G , so that the edge weights w_t implicitly represents d_t . The same idea is used in [19] to efficiently simulate online prediction algorithms with multiplicative update rules, where the decision space is the set of paths of a given directed acyclic graph.

To describe the idea formally, we need some additional notations. Recall that there exists a one-to-one correspondence between the sample S and the set of all root-to-leaf paths \mathcal{P}_G in G . So, we identify a labeled instance $(x_i, y_i) \in S$ with a path $P \in \mathcal{P}_G$, and we will denote the weight for the instance by $d_t(P)$ instead of $d_t(i)$. Furthermore, let \mathcal{P}_G^+ and \mathcal{P}_G^- denote the set of paths that pass through G^+ and the set of paths that pass through G^- , respectively.

Now we give the two conditions C1 and C2 that the edge weights w_t need to satisfy, so as to represent the path distribution d_t .

C1. The edge weights w_t need to satisfy

$$d_t(P) = \prod_{e \in P} w_{t,e}$$

for every path (labeled instance) $P \in \mathcal{P}_G$.

C2. The outflow from each internal node should be one. That is, w_t need to satisfy

$$\sum_{a:(u,a) \in E(G)} w_{t,(u,a)} = 1$$

for every internal node u , where $E(G)$ denotes the set of edges of G .

What we need to show is how to simulate AdaBoost* efficiently by using the edge weights w_t . More precisely, we need to simulate the two parts of AdaBoost*:

- (a) updating the path distributions d_t (corresponding to Line 2 and Line 3-(e) of Algorithm 1), and
- (b) computing the edges $\gamma_{t,j}$ (corresponding to Line 3-(a)).

In the following subsections, we give algorithms that simulate the two parts.

¹ Note that it always holds that $|G| \leq nm$.

Algorithm 2. Initializing the path distribution

1. Let $w'_e = 1$ for all edges in G .
 2. Apply the Weight Pushing algorithm to w' and get w_1 .
-

Algorithm 3. Updating the path distribution

1. For all $e \in E(G)$, let $w'_e = w_{t,e}$.
 2. For all $e \in E(G^+)$ such that $h_{j_t} \in \Phi(e)$, let $w'_e = w'_e \exp(-\alpha_{j_t})$.
 3. For all $e \in E(G^-)$ such that $h_{j_t} \in \Phi(e)$, let $w'_e = w'_e \exp(\alpha_{j_t})$.
 4. Apply the Weight Pushing algorithm to w' and get w_{t+1} .
-

4.1 Updating the Path Distributions d_t

To simulate this part, we use the Weight Pushing algorithm developed by [13], which rearranges the edge weights so that relative weights on the path remain unchanged but again satisfy the two conditions. More precisely, the Weight Pushing algorithm has the following property.

Proposition 4 (Mohri [13]). *When given arbitrary edge weights $w'_e \geq 0$, the Weight Pushing algorithm produces edge weights w_e in time $O(|E|)$ such that w_e satisfies condition C2 and*

$$\prod_{e \in P} w_e = \frac{\prod_{e \in P} w'_e}{\sum_{P \in \mathcal{P}_G} \prod_{e \in P} w'_e}$$

for every path $P \in \mathcal{P}_G$.

The initialization of the path weights ($d_1(P) = 1/m$) of Line 2 of Algorithm 1 can be realized by the two steps as described in Algorithm 2. It is justified by Proposition 4 which implies

$$\prod_{e \in P} w_{1,e} = \frac{1}{|\mathcal{P}_G|} = 1/m = d_1(P).$$

Moreover, the running time of Algorithm 2 is $O(|E|)$.

The update of path distributions of Line 3-(e) of Algorithm 1 can be realized by multiplying the weights of the edges e such that $h_{j_t} \in \Phi(e)$, and applying the Weight Pushing algorithm. See Algorithm 3 for more details.

Below we give a justification of Algorithm 3.

Lemma 5. *Algorithm 3 exactly simulates Line 3-(e) of Algorithm 1 in time $O(|E|)$.*

Proof. Let P be a path in \mathcal{P}_G that corresponds to a labeled instance (x_i, y_i) and examine the quantity $\prod_{e \in P} w'_e$. Recall that

$$\bigcup_{e \in P} \Phi(e) = \{h_j \in H \mid h_j(x_i) = 1\}$$

by the definition of the NZDD construction for S .

First consider the case where $h_{j_t}(x_i) = 0$. In this case, there is no edge $e \in P$ such that $h_{j_t} \in \Phi(e)$. Therefore, $w'_e = w_{t,e}$ for all edges e in P . Thus,

$$\begin{aligned} \prod_{e \in P} w'_e &= \prod_{e \in P} w_{t,e} = d_t(P) \\ &= d_t(P) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)) \\ &= d_t(i) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)). \end{aligned}$$

Next consider the case where $y_i = 1$ (i.e., P passes through G^+) and $h_{j_t}(x_i) = 1$. In this case, there exists a unique edge $e \in P$ such that $h_{j_t} \in \Phi(e)$. The uniqueness comes from Property 3 of the NZDD. So, $w'_e = w_{t,e} \exp(-\alpha_{j_t})$ for the edge e . Since $h_{j_t} \notin \Phi(e')$ for any other edge $e' \in P$, we have

$$\begin{aligned} \prod_{e \in P} w'_e &= \left(\prod_{e \in P} w_{t,e} \right) \exp(-\alpha_{j_t}) \\ &= d_t(P) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)) \\ &= d_t(i) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)). \end{aligned}$$

For the last case where $y_i = -1$ and $h_{j_t}(x_i) = 1$, a similar argument to the case above gives

$$\begin{aligned} \prod_{e \in P} w'_e &= \left(\prod_{e \in P} w_{t,e} \right) \exp(\alpha_{j_t}) \\ &= d_t(P) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)) \\ &= d_t(i) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)). \end{aligned}$$

Hence for all paths P , we have

$$\prod_{e \in P} w'_e = d_t(i) \exp(-\alpha_{j_t} y_i h_{j_t}(x_i)).$$

Therefore, Proposition 4 ensures that w_{t+1} represents the path distribution d_{t+1} as desired. □

4.2 Computing the Edges $\gamma_{t,j}$

To compute $\gamma_{t,j}$, we first compute the following quantity

$$f_e = \sum_{P \in \mathcal{P}_G: e \in P} d_t(P)$$

for all edges e , which can be interpreted as the probability flow of edge e , i.e., the probability that the path P goes through edge e when P is chosen according to the distribution d_t . Since G is a directed acyclic graph, we can compute f_e for all edges e by dynamic programming (e.g., the forward-backward algorithm) in linear time. Then, it is not hard to see that $\gamma_{t,j}$ can be computed by

$$\gamma_{t,j} = \sum_{e \in E(G^+): h_j \in \Phi(e)} f_e - \sum_{e \in E(G^-): h_j \in \Phi(e)} f_e.$$

We summarize the result as in the following lemma.

Lemma 6. *There exists an algorithm that exactly simulates Line 3-(a) of Algorithm 1 in time $O(|G|)$.*

Theorem 2 follows from Lemmas 5 and 6.

5 Conclusions

We have proposed the NZDD, a variant of ZDDs for representing the training data succinctly and algorithms, given a NZDD, simulate AdaBoost* as well as AdaBoost on the training data efficiently. As future work, we will evaluate empirical performances of our method on real and synthetic data sets. Also, investigation of efficient construction methods of NZDDs is important. One of open problems is to extend our method to the 1-norm soft margin maximization, where additional constraints make the theoretical results of the direct application of our method worse. Also, the problem of obtaining similar results for the 2-norm support vector machines remains open.

Acknowledgments. We thank anonymous reviewers for helpful comments. This work is supported in part by JSPS KAKENHI Grant Number JP16J04621, JP16K00305 and JP15H02667, respectively.

References

1. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.* **55**(1), 119–139 (1997)
2. Goto, K., Bannai, H., Inenaga, S., Takeda, M.: Fast q-gram mining on SLP compressed strings. *J. Discrete Algorithms* **18**, 89–99 (2013)
3. Hazan, E.: *Introduction to Online Convex Optimization*. Now Publishers Inc., Hanover (2016)
4. Hermelin, D., Landau, G.M., Landau, S., Weimann, O.: A unified algorithm for accelerating edit-distance computation via text-compression. In: 26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009) (2009)
5. Inoue, T., Takano, K., Watanabe, T., Kawahara, J., Yoshinaka, R., Kishimoto, A., Tsuda, K., Minato, S., Hayashi, Y.: Distribution loss minimization with guaranteed error bound. *IEEE Trans. Smart Grid* **5**(1), 102–111 (2014)
6. Jiang, T., Ravikumar, B.: Minimal NFA problems are hard. *SIAM J. Comput.* **22**(6), 1117–1141 (1993)

7. Knuth, D.E.: Art of Computer Programming. Fascicle 1, The: Bitwise Tricks & Techniques; Binary Decision Diagrams, vol. 4. Addison-Wesley, Reading (2009)
8. Lifshits, Y.: Processing compressed texts: a tractability border. In: Proceedings of the 18th Annual Conference on Combinatorial Pattern Matching, CPM 2007, pp. 228–240 (2007)
9. Mangasarian, O.L.: Arbitrary-norm separating plane. *Oper. Res. Lett.* **24**(1–2), 15–23 (1999)
10. Minato, S.: Zero-suppressed BDDs for set manipulation in combinatorial problems. In: Proceedings of the 30th International Conference on Design Automation, DAC 1993 (1993)
11. Minato, S., Uno, T.: Frequentness-transition queries for distinctive pattern mining from time-segmented databases. In: Proceedings of the 10th SIAM International Conference on Data Mining (SDM 2010), pp. 339–349 (2010)
12. Minato, S., Uno, T., Arimura, H.: LCM over ZBDDs: fast generation of very large-scale frequent itemsets using a compact graph-based representation. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 234–246 (2008)
13. Mohri, M.: General algebraic frameworks and algorithms for shortest-distance problems. Technical report, Technical Memorandum 981210–10TM, AT&T Labs-Research, 62 pages (1998)
14. Nishino, M., Yasuda, N., Minato, S., Nagata, M.: Accelerating graph adjacency matrix multiplications with adjacency forest. In: Proceedings of the 2014 SIAM International Conference on Data Mining (SDM 2014), pp. 1073–1081 (2014)
15. Rätsch, G.: Robust boosting via convex optimization: theory and applications. Ph.D. thesis, University of Potsdam (2001)
16. Rätsch, G., Warmuth, M.K.: Efficient margin maximizing with boosting. *J. Mach. Learn. Res.* **6**, 2131–2152 (2005)
17. Revuz, D.: Minimisation of acyclic deterministic automata in linear time. *Theor. Comput. Sci.* **92**, 181–189 (1992)
18. Tabei, Y., Saigo, H., Yamanishi, Y., Puglisi, S.J.: Scalable partial least squares regression on grammar-compressed data matrices. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2016), pp. 1875–1884 (2016)
19. Takimoto, E., Warmuth, M.: Path kernels and multiplicative updates. *J. Mach. Learn. Res.* **4**, 773–818 (2003)

On Multiple Longest Common Subsequence and Common Motifs with Gaps (Extended Abstract)

Suri Dipannita Sayeed^(✉), M. Sohel Rahman, and Atif Rahman

Department of Computer Science and Engineering,
Bangladesh University of Engineering and Technology, Dhaka 1000, Bangladesh
suri.asha6@gmail.com, {msrahman,atif}@cse.buet.ac.bd

Abstract. Motif finding is the problem of identifying recurring patterns in sequences. It has been widely studied and several variants have been proposed. Here, we address the problem of finding common motifs with gaps that are present in all strings of a finite set. We prove that the problem is NP-hard by reducing the multiple longest common subsequence (MLCS) problem to it. We also provide a branch and bound algorithm for MLCS and show how the algorithm can be extended to give an algorithm for finding common motifs with gaps after common factors that occur in all the strings have been identified.

Keywords: Computational biology · Motif finding · Complexity
Branch and bound

1 Introduction

Motifs are recurring patterns in sequences and motif finding is a widely studied problem in computational biology. It has diverse applications for example in identifying co-expressed genes. Expression of a gene usually requires binding of a transcription factor in the promoter region. Presence of near identical sequences in promoter regions of genes indicates that they are regulated by the same transcription factor and are likely to be co-expressed.

In many cases, the motifs may not be identical and motif finding algorithms need to be robust to differences in sequences. In addition, motifs may not be contiguous. Considering these, the problem of finding common motifs with gaps was introduced [2, 5]. Antoniou et al. gave an algorithm polynomial in length of strings and exponential in number of strings using finite automata and conjectured that asymptotically more efficient algorithms may not be possible [2]. Other variants of the problem with constraints on lengths of gaps have also been proposed and algorithms have been provided [1].

In this paper, we prove that the problem of finding common motif with gaps is NP-hard, for alphabet size of four or more, by reducing to it the multiple longest common subsequence (MLCS) problem, i.e., the problem of finding the longest

common subsequence among a set of sequences. MLCS is also a well studied problem in theoretical computer science and has applications in computational genomics. MLCS was proved NP-hard by Maier [7] and dynamic programming algorithms are known that run in time $O(n^d)$, for d sequences with maximum length, n [6]. Wang et al. [9] gave a dominant point based algorithm with the divide and conquer approach to compute the dominant points and designed a Quick-DP algorithm using those points and later Yang et al. [10] presented a progressive algorithm with efficient parallelization. Huang and Lim gave a branch and bound algorithm using minimum of pairwise longest common subsequence lengths as a bounding condition [4]. As MLCS problem is widely used in protein and genome sequence analysis, further improvement of the algorithm can contribute significantly in the studies of computational genomics [3].

In this paper, first we formally settle the question of complexity of the problem of finding common motifs with gaps. We also present a branch and bound algorithm for computing the longest common subsequences of a set of strings. We pre-process the sequences to explore more promising paths first and to speed-up the pruning process and we explore bounding strategies previously not considered [4]. We then extend this algorithm to find common motifs with gaps. We first find the common factors appearing in all the strings and then use a branch and bound algorithm to chain together the factors. Although this problem is treated as a hard problem in the literature and is handled accordingly, to the best of our knowledge this is the first attempt to prove the hardness thereof formally. And the reduction from MLCS allows us to adapt the branch and bound algorithm for MLCS to solve the problem of finding common motifs with gaps.

The rest of the paper is organized as follows. We give the problem definitions in Sect. 2 and prove the hardness of the problem of finding common motifs with gaps in Sect. 3. The algorithms are presented in Sect. 4 along with examples. Finally, in Sect. 5 we conclude the paper.

2 Background

A motif is a common pattern that appears frequently among a set of genome sequences. In this section we formally define the two problems that we are going to address in the rest of the paper.

2.1 Common Motifs with Gaps

Consider a set of strings $S = \{S_1, S_2, \dots, S_d\}$ over the alphabet $\Sigma = \{A, C, G, T\}$ and two integers p, q , where $1 \leq p \leq q \leq \min(|S_j| : j \in \{1, \dots, d\})$ are given. The problem of finding common motifs with gaps (CMG) aims at finding common words P_1, P_2, \dots, P_m such that: $P_1 *^{d_{i,1}} P_2 *^{d_{i,2}} \dots *^{d_{i,m-1}} P_m$ occurs in S_i , for all $i \in \{1, \dots, d\}$, $m > 1$, $p \leq |P_j| \leq q$ for all $j \in \{1, \dots, m\}$ and $d_{i,j} \geq 1$ for all $i \in \{1, \dots, d\}, j \in \{1, \dots, m-1\}$. Here, $*$ is the don't care symbol that matches any character in Σ .

For example, given a set of three strings $S = S_1, S_2, S_3$ and minimum factor size, $p = 1$ and maximum factor size, $q = 2$, the substrings AC , AA and CA form the common motifs that satisfy the required criteria as highlighted below:

$$\begin{aligned} S_1 &= \mathbf{ACAAAACACAAA} \\ S_2 &= \mathbf{ACACCAACCACA} \\ S_3 &= \mathbf{CACAAACCACCA} \end{aligned}$$

In the optimization version of the problem, we want to maximize m i.e. we seek a common motif with gaps with maximum number of factors and in the decision version of the problem, for a given m , we want to check whether there is a common motif with gaps with $\geq m$ factors.

2.2 Multiple Longest Common Subsequence

The Multiple Longest Common Subsequence (MLCS) problem aims at finding a longest subsequence shared among a set of sequences. Let, $S = \{S_1, S_2, S_3, \dots, S_d\}$ be a set of sequences over a finite alphabet Σ . The Longest Common Subsequence (LCS) of set S is a sequence s with length ℓ , such that it is of the highest length among all subsequences that are shared among all $S_i, i \in \{1, \dots, d\}$. For example,

$$\begin{aligned} S_1 &= \textit{informatics} \\ S_2 &= \textit{bioinformatics} \\ S_3 &= \textit{proteomics} \end{aligned}$$

One of the subsequences for this example is $s_1 = \textit{mics}$, one is $s_2 = \textit{tics}$ and another is $s_3 = \textit{omics}$. The longest one is $s_3 = \textit{omics}$. Notably, this sequence may not necessarily be unique.

In the optimization and decision versions of the problem we intend to find an LCS of the maximum length and decide if there is an LCS greater or equal to a given length, respectively.

3 Complexity of Common Motifs with Gaps

Theorem 1. *CMG problem is NP-complete.*

Proof. To show that the decision version of $CMG \in NP$, for a given set $S = \{S_1, S_2, \dots, S_d\}$ of sequences, a sub-sequence s' of common factors, and an integer $m > 1$, we can easily check in polynomial time whether s' consists of m or more factors and satisfy the length constraints if any, and whether the factors in s' appear in the right order in every sequence of S .

We next prove $MLCS \leq_P CMG$ which shows that CMG is NP-hard.

Given an instance of MLCS over an alphabet Σ given by sequences $T = \{T_1, T_2, \dots, T_d\}$, we construct an instance of CMG, $S = \{S_1, S_2, \dots, S_d\}$ over the alphabet $\{A, C, G, T\}$ such that there exists an LCS of length k for set T if, and only if, S has a CMG of k factors as follows:

1. First we relabel the characters of the MLCS instance using integers from $\{1, \dots, |\Sigma|\}$ and convert each integer into its binary form.
2. We then replace ‘0’s and ‘1’s by ‘A’s and ‘C’s respectively to get strings over $\{A, C\}$ for each integer.
3. Finally we put these strings in the same order in each S_i as the corresponding integers appeared in T_i separated by ‘G’s in S_1 and ‘T’s in all other S_i and we set minimum factor length $p = \lceil \log |\Sigma| \rceil$.

Following is an example of the construction:

a b c d	1 2 3 4	001 010 011 100	AAC G ACA G ACC G CAA
b a c d \Rightarrow	2 1 3 4 \Rightarrow	010 001 011 100 \Rightarrow	ACA T AAC T ACC T CAA
a c b d	1 3 2 4	001 011 010 100	AAC T ACC T ACA T CAA

Now we show that this transformation of T into S is a reduction.

First suppose that T has a solution, that is a sequence t' of k characters are present in every sequence of T in exactly the same order. These characters, i.e., integers in t' will correspond to substrings of each S_i and these substrings will form a sequence s' of k factors. s' is a common motif sequence with k factors since according to the construction the factors must appear in each S_i in exactly the same order and there must be a gap of length at least one between any two factors.

Conversely, suppose, S has a common gapped motif sequence s' with k factors, $k > 1$, that follows a certain order in every sequence S_i . Note that since ‘G’ was used as the separator in S_1 and ‘T’ was used in all other strings, each factor must be strings over $\{A, C\}$ and corresponds to an integer in the LCS instance by construction, and they will follow the same order in every sequence giving us a common subsequence of length k in all strings in T . □

4 Algorithms

Algorithms are known for both MLCS and CMG problems that run in time polynomial in lengths of sequences and exponential in the number of sequences. Complexity results in [7] and in this paper indicate that asymptotically faster algorithms are unlikely. However, search space may be reduced by pruning leading to faster algorithms in practice. Here we present branch and bound algorithms for both MLCS and CMG problems.

4.1 A Branch and Bound Algorithm for MLCS Problem

Given a set of sequences, $S = \{S_1, S_2, \dots, S_d\}$, where $|S_i| \leq n$ for $1 \leq i \leq d$, we preprocess the sequences to explore promising paths first and prune the search space using the value of the best solution found so far and the upper bound on values of solutions the path being explored may lead to. The algorithm uses memoization to avoid redundancy of work, i.e., the values of subproblems already calculated are stored in a table, V indexed by vectors of indices into the sequences.

Preprocessing. We preprocess the sequences to generate candidate lists that will be used to decide in what order nodes are visited during the search process. The candidate list for the first element of the longest common subsequence, C_{init} , consisting of triples $\langle element, minMultiplicity, minDistance \rangle$, is generated as follows:

1. Process each sequence, S_i and list each element e , the distance of its first occurrence to the end of the sequence, $d_{e,i}$ and the number of times it appears in the sequence, $m_{e,i}$.
2. Intersect the lists to get elements common in all sequences. When we intersect we retain the minimum of distances to ends of the sequences for an element, and the minimum multiplicity of the element. Therefore, $minMultiplicity$ and $minDistance$ entries corresponding to $element$, which appears in all the sequences, are given by:

$$minMultiplicity = \min_{1 \leq i \leq d} m_{element,i}$$

$$minDistance = \min_{1 \leq i \leq d} d_{element,i}$$

3. Sort the triples in descending order of minimum distance to ends of sequences and record the sum of multiplicities.

The elements will be explored according to the order in the candidate list, the intuition being an element more distant to the ends of the sequences has more room for other elements to follow it in the LCS.

Similarly, for each element x that appears K_x times in all the sequences, we construct lists $C_{x,k}$ for $1 \leq k \leq K_x$ of triples corresponding to elements that follow the k -th occurrence of x in all the sequences.

For a finite alphabet, each such list can be constructed in time $O(nd)$ and since there can be at most n such lists, preprocessing takes $O(n^2d)$ time.

Branch and Bound. At each node, we take as input a vector of indices $\mathbf{I} = \langle i_1, i_2, \dots, i_d \rangle$ and a common subsequence, α of the sequences $S_1[1 \dots i_1], S_2[1 \dots i_2], \dots, S_d[1 \dots i_d]$, i.e., common subsequence up to \mathbf{I} . We also maintain the best solution found so far globally. We start at $\langle 0, \dots, 0 \rangle$ with the common subsequence ϵ .

Then, at each node, we do the following:

1. Look up the last character and its multiplicity in α and retrieve the corresponding candidate list.
2. Iterate through the $\langle element, minMultiplicity, minDistance \rangle$ triples in the candidate list.
3. Estimate upper bound (see **Pruning conditions** discussed shortly) to check if the branch can be pruned.

4. Find positions $\mathbf{P} = \langle p_1, p_2, \dots, p_d \rangle$ in each sequence following the input indices where *element* occurs. Note that such positions may not exist in some sequences as the k -th occurrence of x in α may correspond to a position in S_i to the right of the position of the k -th occurrence of x in S_i . We skip such entries.
5. If $\langle p_1, p_2, \dots, p_d \rangle$ has already been computed, then look up the value. Otherwise, explore $\langle p_1, p_2, \dots, p_d \rangle$ and update the best solution if needed.

Pruning Conditions. Suppose we are considering for exploration a triple, $\langle element, minMultiplicity, minDistance \rangle$ and suppose this would be the ℓ -th occurrence of *element* in the common subsequence. The following properties can be used to calculate an upper bound on the maximum possible value, \tilde{v} in the subtree rooted at the node:

1. \tilde{v} can not exceed $1 + minDistance$ since there are only $minDistance$ elements after *element* in at least one of the sequences.
2. Similarly, sum of multiplicities of $C_{element, \ell}$ is an upper bound on the number of elements that can follow *element* in the common subsequence.
3. Let $y = element$ and $p_i(y, \ell)$ be the position of the ℓ -th occurrence of y in the i -th sequence. Now $V[p_1(y, \ell), \dots, p_d(y, \ell)]$ is an upper bound on \tilde{v} because the position in S_i that corresponds to the ℓ -th occurrence of y in the common subsequence must be greater than or equal to $p_i(y, \ell)$ for $1 \leq i \leq d$.

The algorithm is summarized in Algorithm 1.

Algorithm 1. MLCS

```

1: Initialize:  $bestSolution \leftarrow 0$ 
2: MLCS-B&B ( $\langle 0, \dots, 0 \rangle, \epsilon$ )
3: procedure MLCS-B&B( $\mathbf{I}, \alpha$ )
4:    $x \leftarrow lastElement(\alpha)$ 
5:    $k \leftarrow multiplicity(\alpha, x)$ 
6:    $v \leftarrow 0$ 
7:   for each  $\langle y, m, d \rangle \in C_{k, x}$  do
8:      $\ell \leftarrow multiplicity(\alpha, y) + 1$ 
9:      $\tilde{v} \leftarrow \min(d + 1, multiplicitySum(C_{y, \ell}) + 1, V[p_1(y, \ell), \dots, p_d(y, \ell)])$ 
10:    if  $\tilde{v} + |\alpha| > bestSolution$  then
11:       $\mathbf{P} \leftarrow getNextPos(\mathbf{I}, y)$ 
12:      if  $\mathbf{P}$  is valid then
13:        if  $V[\mathbf{P}]$  is not null then
14:           $v \leftarrow \max(v, 1 + V[\mathbf{P}])$ 
15:        else
16:           $v \leftarrow \max(v, 1 + MLCS-B\&B(\mathbf{P}, \alpha.y))$ 
17:    if  $v + |\alpha| > bestSolution$  then
18:       $bestSolution \leftarrow v + |\alpha|$ 
19:     $V[\mathbf{I}] \leftarrow v$ 
20:    Return  $v$ 

```

An Illustrative Example. A simulation of the algorithm on an example is shown in Fig. 1.

Input: $S_1 : ABCXBCYZ$ $S_2 : ABXYCZXCBC$ $S_3 : ABCXYBCZBC$ $S_4 : ABXXCCYZBC$	Preprocessing: $C_{init} : < A, 1, 7 >, < B, 2, 6 >, < C, 2, 4 >, < X, 1, 4 >, < Y, 1, 1 >, < Z, 1, 0 >$ $C_{A,1} : < B, 2, 6 >, < C, 2, 4 >, < X, 1, 4 >, < Y, 1, 1 >, < Z, 1, 0 >$ $C_{B,1} : < C, 2, 4 >, < X, 1, 4 >, < Y, 1, 1 >, < B, 1, 1 >, < Z, 1, 0 >$ $C_{X,1} : < C, 1, 2 >, < Y, 1, 1 >, < B, 1, 1 >, < Z, 1, 0 >$ $C_{C,1} : < B, 1, 1 >, < Z, 1, 0 >, < C, 1, 0 >$ $C_{B,2} : < C, 1, 0 >$ $C_{Y,1} : < Z, 1, 0 >$	$C_{Z,1} = \{ \}$ $C_{C,2} = \{ \}$
--	--	--

Branch and bound:

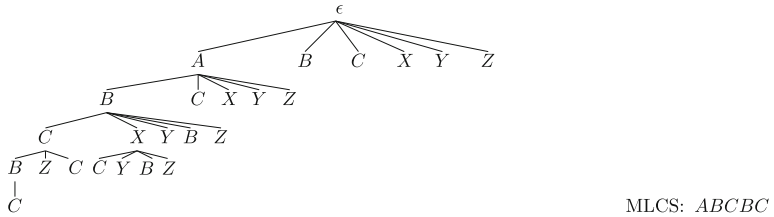


Fig. 1. Simulation of Algorithm 1 on an example set

4.2 A Branch and Bound Algorithm for CMG

We now extend the method discussed above and present a branch and bound algorithm for finding common motifs with gaps (CMG). We are given a set of strings, $S = \{S_1, S_2, \dots, S_d\}$, and integers p, q giving upper and lower bounds on factor lengths respectively.

The first step is to find common factors. Then we use an approach similar to the one for finding MLCS taking into account that there may be multiple factors overlapping a position in a string, at most one of which can be present in the final solution.

Identifying Common Factors. We first identify common factors, i.e., substrings with lengths between p and q that appear in all the strings and record start indices (and end indices implicitly) of their occurrences in every string. This can be done efficiently using approaches such as suffix trees [8], finite automata [2]. For each string S_i , where $1 \leq i \leq d$, we create a list of factors, F_i consisting of all occurrences of all the common factors in the string sorted in ascending order of their end indices.

Candidate List Generation. The factor lists are then processed to generate candidate lists in a similar approach to the one used for preprocessing MLCS instances. In this context, a factor, f_1 will be in the list of candidates to follow

the k -th occurrence of factor f_2 if in each factor list there is an entry for f_1 with start index exceeding the end index of k -th occurrence of factor f_2 by at least 2. The candidate lists are sorted in descending order of minimum distances of factors from the ends of factor lists.

Branch and Bound and Pruning Conditions. The branch and bound algorithm now proceeds as the one for finding MLCS producing a common motif with highest number of factors.

An Illustrative Example. Figure 2 shows simulation of the algorithm for finding common motifs with gaps on a sample instance. The factor lists consist of pairs denoting the factor string and its start position sorted in increasing order of their end positions i.e. sum of the start positions and the lengths of factors. Note that although TCG follows TG in each of the factor lists, it is not included in the list of candidates to follow the first occurrence of TG since it overlaps with TG in two of the strings. We also see that although TGC is a common factor of the strings, it does not appear in the final common motif with gap as that would lead a motif with only one factor. However, substrings of TGC are considered as factors - TG in S_1 and S_3 and GC in S_2 - to obtain a common motif with three factors.

<p>Input:</p> <p>$S_1 : GCGCTGCACTG$</p> <p>$S_2 : TGCTATGTCTGT$</p> <p>$S_3 : GGCATGAATGC$</p> <p>$p = 2, q = 3$</p>	<p>Factor lists:</p> <p>$F_1 : < GC, 1 >, < GC, 3 >, < TG, 5 >, < TGC, 5 >, < GC, 6 >, < TG, 10 >$</p> <p>$F_2 : < TG, 1 >, < TGC, 1 >, < GC, 2 >, < TG, 6 >, < TG, 10 >$</p> <p>$F_3 : < GC, 2 >, < TG, 5 >, < TG, 9 >, < TGC, 9 >, < GC, 10 >$</p>
---	---

Candidate Lists:

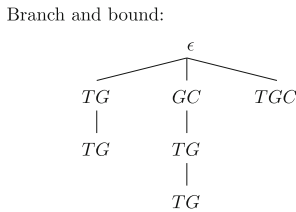
$C_{init} : < TG, 2, 3 >, < GC, 1, 2 >, < TGC, 1, 1 >$

$C_{TG,1} : < TG, 1, 0 >$

$C_{GC,1} : < TG, 2, 1 >$

$C_{TGC,1} : \{ \}$

$C_{TG,2} : \{ \}$



CMG: $GC-TG-TG$

Fig. 2. Simulation of the algorithm for finding common motifs with gaps on an example

5 Conclusions

In this paper, we have addressed two problems with applications in genomics - finding a longest common subsequence of multiple sequences (MLCS) and finding common motifs with gaps (CMG). MLCS is known to be NP-hard and its reduction to CMG in this paper formally proves that CMG is NP-hard as well.

While this makes polynomial time algorithms for the problems unlikely, we have proposed a branch and bound algorithm with preprocessing to prune the search space that may reduce running time for many instances of the problems. Future work will include implementation of the algorithms to test the speed-up achieved compared to existing algorithms of the problems and subsequent application to real datasets.

References

1. Antoniou, P., Crochemore, M., Iliopoulos, C., Peterlongo, P.: Application of suffix trees for the acquisition of common motifs with gaps in a set of strings. In: International Conference on Language and Automata Theory and Applications (2007)
2. Antoniou, P., Holub, J., Iliopoulos, C.S., Melichar, B., Peterlongo, P.: Finding common motifs with gaps using finite automata. In: Ibarra, O.H., Yen, H.-C. (eds.) CIAA 2006. LNCS, vol. 4094, pp. 69–77. Springer, Heidelberg (2006). https://doi.org/10.1007/11812128_8
3. Chen, Y., Wan, A., Liu, W.: A fast parallel algorithm for finding the longest common sequence of multiple biosequences. *BMC Bioinformatics* **7**(4), S4 (2006)
4. Huang, G., Lim, A.: An effective branch-and-bound algorithm to solve the k-longest common subsequence problem. In: Proceedings of the 16th European Conference on Artificial Intelligence, pp. 191–195. IOS Press (2004)
5. Iliopoulos, C.S., McHugh, J., Peterlongo, P., Pisanti, N., Rytter, W., Sagot, M.F.: A first approach to finding common motifs with gaps. *Int. J. Found. Comput. Sci.* **16**(06), 1145–1154 (2005)
6. Korkin, D., Wang, Q., Shang, Y.: An efficient parallel algorithm for the multiple longest common subsequence (MLCS) problem. In: 2008 37th International Conference on Parallel Processing, ICPP 2008, pp. 354–363. IEEE (2008)
7. Maier, D.: The complexity of some problems on subsequences and supersequences. *J. ACM (JACM)* **25**(2), 322–336 (1978)
8. Marsan, L., Sagot, M.F.: Extracting structured motifs using a suffix tree algorithms and application to promoter consensus identification. In: Proceedings of the Fourth Annual International Conference on Computational Molecular Biology, pp. 210–219. ACM (2000)
9. Wang, Q., Korkin, D., Shang, Y.: A fast multiple longest common subsequence (MLCS) algorithm. *IEEE Trans. Knowl. Data Eng.* **23**(3), 321–334 (2011)
10. Yang, J., Xu, Y., Sun, G., Shang, Y.: A new progressive algorithm for a multiple longest common subsequences problem and its efficient parallelization. *IEEE Trans. Parallel Distrib. Syst.* **24**(5), 862–870 (2013)

FPT Algorithms Exploiting Carving Decomposition for Eulerian Orientations and Ice-Type Models

Shinya Shiroshita^(✉), Tomoaki Ogasawara, Hidefumi Hiraishi,
and Hiroshi Imai

Graduate School of Information Science and Technology, The University of Tokyo,
7-3-1 Hongo, Bunkyo-ku, Tokyo, Japan
{castle_phidnight,t.ogasawara,hiraishi1729,imai}@is.s.u-tokyo.ac.jp

Abstract. An Eulerian orientation of an undirected graph is an orientation of edges such that, for each vertex, both the indegree and the outdegree are the same. Eulerian orientations are important in a variety of fields. In statistical physics, the partition function of the so-called ice model, which is the special case of the ice-type model, is related to the number of Eulerian orientations of a 4-regular graph, which is the value of its Tutte polynomial at the point $(0, -2)$. The problem of counting the number of Eulerian orientations in a 4-regular graph is #P-complete, and yet there is an FPT (Fixed Parameter Tractable) algorithm for it with respect to the tree-width of the graph.

This paper presents two FPT algorithms based on a carving decomposition. One of them counts the number of Eulerian orientations for a general graph in $O(k \cdot (2\sqrt{2})^k \cdot n)$ time and $O(2^k \cdot n)$ memory consumption, and the other calculates the partition function of a general ice-type model for a 4-regular graph in $O(k \cdot (2\sqrt{2})^k \cdot n)$ time and $O(2^k \cdot n + (2\sqrt{2})^k)$ memory consumption where, for an input graph, k is the carving-width and n is the size of the vertex set.

1 Introduction

An *Eulerian circuit* for an undirected graph is a closed walk which visits every edge of a graph exactly once. For the problem of finding an Eulerian circuit, the solution that Euler showed is considered as one of the origins of graph theory [8]. An *Eulerian orientation* of an undirected graph G is an orientation of edges such that, for each vertex, both the indegree and the outdegree are the same. It is known that G has an Eulerian orientation if and only if G has an Eulerian circuit.

Eulerian orientations have various significances in several fields. In statistical physics, Pauling [16] introduced the so-called ice problem on a toroidal square lattice, and derived a rough value about its physical quantity in terms of the number of ‘ice configurations’. Lieb [12] presented its exact value, thus resolving the case of square lattices, and opening research on other general cases. Welsh [24] pointed out that, for a 4-regular planar graph, the number of ice configurations is the number of Eulerian orientations, which is equal to the value of

Tutte polynomial at the point $(0, -2)$. This holds for a general 4-regular graph via nowhere zero 3-flows [25]. Moreover, Welsh also referred to investigating the #P-completeness of these problems in connection with the #P-completeness of evaluating the Tutte polynomial of a graph. In graph theory, motivated by a problem of evaluating the permanent of doubly stochastic matrices and the $(0,1)$ -matrix posed by Minc [14] and van der Waerden [23], Schrijver gave the upper and lower bounds of the number of Eulerian orientations [18].

An Eulerian orientation of a graph can be computed in polynomial time. However, computing the number of Eulerian orientations is not easy. Mihail and Winkler [13] showed that computing the number of Eulerian orientations is #P-complete in a general graph, while there exists an FPRAS for this problem. The problem for a planar graph was shown to remain #P-complete [7]. Valiant [22] introduced holographic algorithms in connection with quantum computing, and proved that there exists a polynomial-time algorithm that computes the number of orientations of a 3-regular graph such that, for each vertex, both the indegree and the outdegree are less than three. Huang and Lu [11] advanced research on the notion of holographic algorithms and showed that, for $2k$ -regular graphs, counting the number of Eulerian orientations is #P-complete for any $k \geq 2$.

The problem of counting the number of Eulerian circuits is similar to that for orientations. Brighton and Winkler [3] presented the proof that the counting Eulerian tours is #P-complete. Even for a 4-regular planar graph, the counting Eulerian tours remains #P-complete [10]. Chebolu et al. [6] claimed that the counting Eulerian tours can be solved in polynomial time on bounded tree-width, and then Balaji et al. [2] proved that there exists a polynomial-time algorithm for counting Eulerian tours on bounded tree-width (efficiently even in parallel computing).

In recent years, for such an intractable problem, there exist many studies that measure the complexity of a problem with respect to parameters and the input size. A problem is said to be *FPT* (Fixed Parameter Tractable) if the problem can be solved in $f(k) \cdot n^{O(1)}$ time where $f(k)$ is an arbitrary function in a parameter k and n is the input size. An FPT algorithm, with respect to the path-width, of computing the Tutte polynomial was constructed by Sekine et al. [19]. Andrzejak [1] and Noble [15] independently showed that there exists a polynomial-time algorithm for computing the Tutte polynomial in graphs of bounded tree-width, and the algorithm in the latter paper is FPT. By this existence of an FPT algorithm which computes the Tutte polynomial, we consider that there exists an FPT algorithm on tree-width that counts the number of Eulerian orientations. With respect to the research of Sasák [17] that tree-width is bounded by carving-width, Noble's algorithm is also an FPT on carving-width. However, this algorithm has the time complexity at least the square of the input size.

In this paper, we propose an FPT algorithm for counting Eulerian orientations in a general graph by exploiting a carving-width as a parameter, and runs in $O(k \cdot (2\sqrt{2})^k \cdot n)$ time and $O(2^k \cdot n)$ memory where, for an input general graph, k is the carving-width and n is the size of the vertex set. Our algorithm is based



Fig. 1. Configurations of the ice-type model.

on dynamic programming on a binary tree corresponding to the decomposition, which recursively determines orientations of the subgraph induced by each node of the tree. We use a linear-time algorithm of Thilikos et al. [21] to construct a carving decomposition of any undirected graph whose width is not more than a given parameter k .

In Sect. 4, we focus on the partition function of the ice-type model (or the six-vertex model) introduced by Lieb [12], which is the generalized version of the ice problem. The model gets six parameters describing the weight of each configuration, which affect the weight of each Eulerian orientation. Figure 1 shows six possible configurations of each vertex. From the viewpoint of computational complexity, Cai et al. [5] divided parameters into two classes: calculating their partition function can be done in polynomial time, or it is #P-hard. Cai et al. [4] categorized parameters into three cases: polynomial time in general graphs, polynomial time in planar graphs, or #P-hard even in planar graphs. This paper provides another FPT algorithm which evaluates the partition function of ice-type models for any 4-regular graph in $O(k \cdot (2\sqrt{2})^k \cdot n)$ time and $O(2^k \cdot n + (2\sqrt{2})^k)$ memory consumption.

2 Definitions

2.1 Graph Definitions

Let $G = (V, E)$ be a graph (can be undirected or directed), and let $V(G)$ ($E(G)$) be the vertex (edge) set of G . We denote as $G[V']$ ($V' \subseteq V$) the subgraph of G induced by V' . Each edge connects two different vertices. There may exist a set of two vertices which has more than one edge connecting them.

For a directed graph, we call a directed edge as an *arc*. If an arc is directed from a vertex y to a vertex x , then x is called the head and y is called the tail. For each vertex v , the *indegree* $d_{in}(v)$ is the number of arcs whose head is v , and the *outdegree* $d_{out}(v)$ is the number of arcs whose tail is v .

An *Eulerian circuit* for an undirected graph is a closed walk which visits each edge of G exactly once. An *Eulerian orientation* of an undirected graph G is an orientation of all edges such that, for each vertex v , both $d_{in}(v)$ and $d_{out}(v)$ are the same.

We also define a *cut* of a graph. For a graph $G = (V, E)$, a cut of G is a partition of V into two vertex set (A, B) . The *cut-set* of a cut (A, B) is the set of edges crossing between A and B . The size of a cut is defined as the number of edges in its cut-set.

2.2 Carving Decomposition

A *carving decomposition* of a graph $G = (V, E)$ is an unrooted binary tree T whose leaves correspond to the vertices of G . For any edge e of T , the graph $T - e$ is consisted of exactly two connected components. Let $(X_e, V \setminus X_e)$ be a cut where $X_e \subseteq V$ and for any vertices v, w whose corresponding leaves are in the same component of $T - e$, either $v, w \in X_e$ or $v, w \in V \setminus X_e$ holds. The *width* of a carving decomposition T is the maximum size of a cut by X_e and $V \setminus X_e$ of all $e \in E(T)$, and the *carving-width* of G is the minimum width of all carving decompositions of G . A carving-width was introduced by Seymour and Thomas [20]. It is NP-complete that deciding whether the carving-width of a graph is at most k .

We assume that a carving decomposition of an input graph is already computed by using the algorithm of Thilikos et al. [21]. In order to use dynamic programming, we assume that the decomposition is given as a rooted binary tree, which is generated from an unrooted binary tree by adding a root node on some edge and directions on every edge.

3 Algorithm for Eulerian Orientations

This section describes our FPT algorithm counting the number of Eulerian orientations. We call it Algorithm1 for convenience. It is based on dynamic programming on a carving decomposition. Algorithm1 gets an input graph $G = (V, E)$ of n vertices and m edges and its carving decomposition as a rooted binary tree whose width is k .

Algorithm1 recursively evaluates the number of orientations of the induced subgraph of G on each node of the decomposition. One important aspect is that once two different orientations of the same subgraph have the same vector of $d_{in} - d_{out}$ of n vertices, we have no necessity of distinguishing them on parent nodes. So we only need to calculate the number of possible orientations for each combination of $d_{in} - d_{out}$ of n vertices. Algorithm1 focuses on only a subset of the combinations for each node S , denoted by \mathcal{D}_S , in order to bound the number of operations on each node by a function of k . Figure 2 shows the pseudo-code of Algorithm1. Time complexity is $O(k \cdot (2\sqrt{2})^k \cdot n)$ and memory complexity is $O(2^k \cdot n)$.

3.1 Description of Algorithm1

This subsection introduces some additional definitions for the latter subsections.

For convenience, we assume that each vertex is numbered from v_1 to v_n . We denote each node of the decomposition as the vertex set it contains. We define the *degree vector* $dvec_A = (d_{in}(v_1) - d_{out}(v_1), d_{in}(v_2) - d_{out}(v_2), \dots, d_{in}(v_n) - d_{out}(v_n))$ of the set of arcs A which is a vector of n integers representing $d_{in} - d_{out}$ of each vertex.

```

RecursivePartOfAlgorithm1(Node S)
  For each  $\sigma \in \mathcal{D}_S$ 
     $dp_S[\sigma] \leftarrow 0$ 
  If  $S$  is a leaf node then
     $dp_S[(0, 0, \dots, 0)] \leftarrow 1$ 
  Else
    Let  $L, R$  be the left/right child of  $S$  respectively.
     $dp_L = \text{RecursivePartOfAlgorithm1}(L)$ 
     $dp_R = \text{RecursivePartOfAlgorithm1}(R)$ 
    For each orientation  $X$  of edges  $E_{L,R}$ 
      For each  $\sigma \in \mathcal{D}_S$ 
         $dp_S[\sigma] \leftarrow dp_S[\sigma] + dp_L[(\sigma - dvec_X) \circ e_L] + dp_R[(\sigma - dvec_X) \circ e_R]$ 
  Return  $dp_S$ 
Algorithm1(Graph G, Carving T)
  Let  $top$  be the top node of  $T$ .
   $dp = \text{RecursivePartOfAlgorithm1}(top)$ .
  Print  $dp[(0, 0, \dots, 0)]$  as an output.

```

Fig. 2. The pseudo-code of Algorithm1.

Let $E_{A,B}$ of two disjoint vertex sets $A, B \subseteq V$ be a set of edges which expand from one set to the other. For any subset $S \subseteq V$ and any vertex $v \in S$, let $B_S(v) = E_{\{v\}, V \setminus S}$ be a set of edges connecting v and a vertex in $V \setminus S$, and let $\mathcal{B}_S(v_i) = \{-|B_S(v_i)|, -|B_S(v_i)| + 2, \dots, |B_S(v_i)|\}$ be the set of integers that $d_{in}(v_i) - d_{out}(v_i)$ must be on the node S . $\mathcal{D}_S = \{(a_1, a_2, \dots, a_n) | a_i \in \mathcal{B}_S(v_i) \text{ for } 1 \leq i \leq n\}$ is a set of n -element vectors we consider on node S . The following Lemma 1 shows that we do not have to consider any degree vector not in \mathcal{D}_S .

Lemma 1. *For any Eulerian orientation O of $E(G)$ and any vertex set $S \subseteq V$, let O_S be the orientation of $E(G[S])$ satisfying $O_S \subseteq O$. Then, $dvec_{O_S} \in \mathcal{D}_S$.*

Proof. If $dvec_{O_S} \notin \mathcal{D}_S$, there is a vertex v such that $d_{in}(v) - d_{out}(v) \notin \mathcal{B}_S(v)$. By the similarity, we may assume that $d_{in}(v) - d_{out}(v) \geq |B_S(v)| + 1$. In order to make $d_{in}(v) - d_{out}(v) = 0$ on O , there must be at least $|B_S(v)| + 1$ out-edges adjacent to v in $O \setminus O_S$. However, there are only $|B_S(v)|$ edges adjacent to v in $E \setminus E(G[S])$. □

Lemma 2 bounds $|\mathcal{D}_S|$ by the size of the cut $(S, V \setminus S)$, which is less than or equal to the carving-width.

Lemma 2. $|\mathcal{D}_S| \leq 2^{E_{S, V \setminus S}}$ for any vertex set $S \subseteq V$.

Proof. For any sequence of nonnegative integers $a_1, a_2, \dots, a_{n'}$, we define

$$\Pi_+(a_1, a_2, \dots, a_{n'}) = \prod_{i=1}^{n'} (a_i + 1).$$

By the definition of \mathcal{D}_S , $|\mathcal{D}_S| = \prod_+(|B_S(v_1)|, |B_S(v_2)|, \dots, |B_S(v_n)|)$.

Since $\sum_{i=1}^n |B_S(v_i)| = E_{S, V \setminus S}$,

$$\begin{aligned} |\mathcal{D}_S| &\leq \max_{a_1, a_2, \dots, a_n \in \mathbb{Z}_{\geq 0}, \sum_{i=1}^n a_i = E_{S, V \setminus S}} \prod_+(a_1, a_2, \dots, a_n) \\ &\leq \max_{n' \in \mathbb{Z}^+} \left\{ \max_{a_1, a_2, \dots, a_{n'} \in \mathbb{Z}_{\geq 0}, \sum_{i=1}^{n'} a_i = E_{S, V \setminus S}} \prod_+(a_1, a_2, \dots, a_{n'}) \right\} \end{aligned}$$

For any sequence $a_1, a_2, \dots, a_{n'}$ which satisfies $a_k \geq 2$ for some $1 \leq k \leq n'$,

$$\begin{aligned} \prod_+(a_1, a_2, \dots, a_{n'}) &< \prod_+(a_1, a_2, \dots, a_{k-1}) \cdot a_k \cdot 2 \cdot \prod_+(a_{k+1}, a_{k+2}, \dots, a_{n'}) \\ \therefore \prod_+(a_1, a_2, \dots, a_{n'}) &< \prod_+(a_1, a_2, \dots, a_{k-1}, a_k - 1, 1, a_{k+1}, a_{k+2}, \dots, a_{n'}). \end{aligned}$$

So we need not consider any sequence containing an integer larger than 1.

Every sequence $a_1, a_2, \dots, a_{n'}$ containing only 0 or 1 satisfies $\prod_+(a_1, a_2, \dots, a_{n'}) = 2^{E_{S, V \setminus S}}$. Therefore, $|\mathcal{D}_S| \leq 2^{E_{S, V \setminus S}}$. \square

We use a 0–1 vector $e_S = \{(a_{v_1}, a_{v_2}, \dots, a_{v_n}) \mid a_{v_i} = 1 \text{ if } v_i \in S \text{ and } a_{v_i} = 0 \text{ otherwise}\}$ to describe the subset $S \subseteq V$. We also denote the operator $x \circ y$ as the Hadamard product of n -element vectors x, y . That is, $(x_1, x_2, \dots, x_n) \circ (y_1, y_2, \dots, y_n) = (x_1 y_1, x_2 y_2, \dots, x_n y_n)$.

3.2 Correctness and Complexity

This subsection describes correctness and time/memory complexity of Algorithm 1.

Theorem 1. *Algorithm 1 correctly counts the number of Eulerian orientations.*

Proof. Before showing Theorem 1, we first prove the following lemma:

Lemma 3. *For any internal node S , its children L, R and orientation X of $E_{L, R}$, let f be any function which gets $v \in S$ and a directed edge set O and returns 0 or 1 depending only on the directions of v 's adjacent edges in O . Let $A_{S', X}$ be the orientation set of $G[S']$ containing X . Let $P_S = \{O \mid O \in A_{S', X}, f(v, O) = 1 \text{ for all } v \in S'\}$. Let $P_C (C \in \{L, R\})$ be the set of an orientation O' of $E(G[C])$ satisfying $f(v, O' \cup X) = 1$ for all $v \in C$. Then, $P_S = \{p \cup X \cup q \mid p \in P_L, q \in P_R\}$.*

Proof (Lemma 3). We will prove both inclusion relations.

\subseteq For any $O \in P_S$, let O_L be a subset of O included in $E(G[L])$. For any $v \in L$, the combination of directions of adjacent edges of v on $O_L \cup X$ is equal to that of O . Therefore, $O_L \in P_L$. It is also true for R .

\supseteq For any pair of $O_L \in P_L$ and $O_R \in P_R$ and for any vertex $v \in S$, the directions of adjacent edges of v on $O_L \cup X \cup O_R$ is equal to those of $O_L \cup X (v \in L)$ or $O_R \cup X (v \in R)$ because no vertex contain both edges of O_L and O_R . \square

For each node S and degree vector $\sigma \in \mathcal{D}_S$, let $A_{S,\sigma}$ be the set of orientations of $E(G[S])$ whose degree vector is equal to σ . We prove $|A_{S,\sigma}| = dp_S[\sigma]$ by the following induction:

When S is a leaf node, $\mathcal{D}_S = \{(0, 0, \dots, 0)\}$ and $A_{S,(0,0,\dots,0)} = \{\emptyset\}$. Therefore $|A_{S,(0,0,\dots,0)}| = 1 = dp_S[(0, 0, \dots, 0)]$.

When S is an internal node and its children L, R satisfy the induction hypothesis, let $A'_{X,S,\sigma}$ be a set of orientations of $E(G[S])$ containing an orientation X of $E_{L,R}$. By Lemma 3, $A'_{X,S,\sigma} = \{p \cup X \cup q \mid p \in A_{L,(\sigma - dvec_X) \circ e_L}, q \in A_{R,(\sigma - dvec_X) \circ e_R}\}$ because the constraint of the Eulerian orientation depends only on $d_{in} - d_{out}$ of each vertex. Therefore,

$$\begin{aligned} dp_S[\sigma] &= \sum_X |A'_{X,S,\sigma}| \\ &= \sum_X (|A_{L,(\sigma - dvec_X) \circ e_L}| \cdot |A_{R,(\sigma - dvec_X) \circ e_R}|) \\ &= \sum_X (dp_L[(\sigma - dvec_X) \circ e_L] \cdot dp_R[(\sigma - dvec_X) \circ e_R]). \end{aligned}$$

□

Theorem 2. *Algorithm1 has $O(k \cdot (2\sqrt{2})^k \cdot n)$ time complexity where k is the carving-width of the input graph.*

Proof. We show that our algorithm calculates all elements of the dynamic programming tables of all nodes on the subtree of the decomposition rooted by S in $O(k \cdot (2\sqrt{2})^k \cdot n_S)$ time where n_S is the number of nodes on the subtree.

When S is a leaf node, our algorithm returns dp_S in constant time.

When S is an internal node and its children L, R satisfy the induction hypothesis, running time of RecursivePartOfAlgorithm1 are bounded by $O(k \cdot (2\sqrt{2})^k \cdot n_S)$ respectively. The total number of operations accessing to one element of dp_L on RecursivePartOfAlgorithm1(S) is equal to $O(|\mathcal{D}_S| \cdot 2^{E_{L,R}})$. Accessing a value in dp_L costs $O(k)$ time because we can calculate an index from at most k integers. The above argument is also correct for dp_R . By Lemma 2, $|\mathcal{D}_S| \leq 2^{E_{L,V \setminus S} + E_{R,V \setminus S}}$. They mean that the calculating time of all elements of dp_S is bounded by $O(k \cdot 2^{E_{L,V \setminus S} + E_{R,V \setminus S} + E_{L,R}})$. By the definition of k , $E_{L,V \setminus S} + E_{R,V \setminus S} \leq k$, $E_{L,V \setminus S} + E_{L,R} \leq k$, and $E_{R,V \setminus S} + E_{L,R} \leq k$ are satisfied. By gathering them, $E_{L,V \setminus S} + E_{R,V \setminus S} + E_{L,R} \leq (3/2)k$. It means that the calculation time is bounded by $O(k \cdot 2^{(3/2)k}) = O(k \cdot (2\sqrt{2})^k)$.

Since the number of nodes on the decomposition is $O(n)$, the whole time complexity is $O(k \cdot (2\sqrt{2})^k \cdot n)$. □

Theorem 3. *Algorithm1 has $O(2^k \cdot n)$ memory complexity where k is the carving-width of the input graph.*

Proof. By Lemma 2, the number of elements on dp_S is bounded by $O(2^k)$ for any node S of the decomposition. □

4 Algorithm for Ice-Type Models

This section describes the other algorithm, denoted as Algorithm2, which calculates the partition function of an ice-type model of a 4-regular graph. Like Algorithm1, Algorithm2 uses the carving decomposition and treats Eulerian orientations on each node of the tree. While Algorithm1 stores the number of orientations, Algorithm2 stores the summation of the current partition function of each orientation on the dynamic programming table. Algorithm2 uses vectors slightly different from degree vectors as the key of the table.

Figure 3 shows the pseudo-code of Algorithm2. We can calculate the given 4-regular graph's partition function in $O(k \cdot (2\sqrt{2})^k \cdot n)$ time consumption and $O(2^k \cdot n + (2\sqrt{2})^k)$ memory consumption. In the pseudo-code, we define that a state vector σ of S does not contradict to an orientation X of $E_{L,R}$ if and only if there exists an orientation O of S such that O contains X and $svec_O = \sigma$.

4.1 Description of Algorithm2

An ice-type model is a pair of a 4-regular graph and six parameters $\epsilon_1, \epsilon_2, \dots, \epsilon_6$ denoting the energy of each configuration. The partition function Z of the model is equal to $\sum_X e^{-(n_1\epsilon_1+n_2\epsilon_2+\dots+n_6\epsilon_6)/k_B T}$ where X is an Eulerian orientation of an input graph, n_i is the number of vertices on X forming i -th configuration, k_B is Boltzmann's constant and T is the temperature of the system.

```

RecursivePartOfAlgorithm2(Node S)
  For each  $\sigma \in \mathcal{S}_S$ 
     $dp_S[\sigma] \leftarrow 0$ 
  If  $S$  is a leaf node then
     $dp_S[(0, 0, \dots, 0)] \leftarrow 1$ 
  Else
    For  $C \in \{L, R\}$  be the left/right child of  $S$  respectively.
      For each  $\sigma_C \in \mathcal{S}_C$  and  $\sigma_C^* \in CV(C, \sigma_C, E_{L,R})$ 
         $dp'_C[\sigma_C^*] \leftarrow 0$ 
         $dp_C = \text{RecursivePartOfAlgorithm2}(C)$ 
        For each  $\sigma_C \in \mathcal{S}_C$  and  $\sigma_C^* \in CV(C, \sigma_C, E_{L,R})$ 
           $dp'_C[\sigma_C^*] \leftarrow dp'_C[\sigma_C^*] + dp_C[\sigma_C] \times CZ(\sigma_C^*)$ 
        For each orientation  $X$  of edges  $E_{L,R}$ 
          For each  $\sigma \in \mathcal{S}_S$  which does not contradict to  $X$ 
             $dp_S[\sigma] \leftarrow dp_S[\sigma] + dp'_L[RVC(S, \sigma, X) \circ e_L] \times dp'_R[RVC(S, \sigma, X) \circ e_R]$ 
    Return  $dp_S$ 
Algorithm2(Graph G, Carving T)
  Let  $top$  be the top node of  $T$ .
   $dp = \text{RecursivePartOfAlgorithm2}(top)$ .
  Print  $dp[(0, 0, \dots, 0)]$  as an output.

```

Fig. 3. The pseudo-code of Algorithm2.

On a subgraph, there exists a vertex the directions of whose adjacent edges are partially undecided. We apply the energy of its configuration to current partition function when three of the four edges' orientation are fixed.

For each vertex v , we have to remember the direction of adjacent edges to decide the configuration. Instead of the degree vector, we use alternative n -element vector *state vector* denoted by $svec$. For any orientation A , $svec_A$ is an n -element vector whose i -th element explains the status of v_i by an integer.

Let $d_S(v)$ be the number of edges on a node S incident to v . We use the following strategy: storing a pair of directions of adjacent edges if $d_S(v) = 2$ and $d_{in} - d_{out}$ otherwise. Our idea is that, when $d_S(v) < 2$, we can determine the direction by $d_{in} - d_{out}$ and when $d_S(v) > 2$, we need not know the direction since the energy of v has already been applied. Let $\mathcal{B}'_S(v) = \{1, 2, 3, 4\}$ if $d_S(v) = 2$ and $\mathcal{B}'_S(v) = \mathcal{B}_S(v)$ otherwise. Let $\mathcal{S}_S = \{(a_1, a_2, \dots, a_n) | a_i \in \mathcal{B}'_S(v_i) \text{ for } 1 \leq i \leq n\}$ be a state vector set we consider on node S . Lemma 4 bounds the size of \mathcal{S}_S .

Lemma 4. $|\mathcal{S}_S| \leq 2^{E_{S, V \setminus S}}$ for any node S .

Proof. Let a_i be the number of vertices whose d_S is i . Since $|\mathcal{B}'_S(v)| \leq 2$ for any v satisfying $d_S(v) \neq 2$, $|\mathcal{S}_S| \leq 2^{a_0 + a_1 + a_3 + a_4} \cdot 4^{a_2} = 2^{E_{S, V \setminus S} - 2a_2} \cdot 4^{a_2} = 2^{E_{S, V \setminus S}}$. \square

In order to reduce time complexity, Algorithm2 uses a *converted vector*, which is a pair of a state vector and *tags*. A converted vector $cvec_{O, X}$ is defined on a directed edge set O and its subset X . Its state vector is equal to $svec_O$. A tag is a 0–1 bit attached to each vertex with exactly two adjacent edges in $O \setminus X$ and at least one adjacent edge in X . We denote such a vertex as an *exceptional vertex*. Each tag explains the direction of the adjacent edge in X (if there exist two adjacent edges in X , the tag focuses on the smaller index one). Note that the tag uniquely determines the configuration of the corresponding vertex.

Algorithm2 uses three functions: CV, CZ and RCV. CV(node S , state vector σ_S of S , edge set E') returns a set of converted vectors $\{cvec_{O \cup X, X} | O, X \text{ is an orientation of } E(G[S]), E' \text{ respectively, } svec_O = \sigma\}$. Since S contains at most $k/2$ vertices whose d_S is two, $|\text{CV}(S, \sigma_S, E')| \leq 2^{k/2} = \sqrt{2}^k$. CZ(converted vector σ^*) returns a correction value of the conversion. Let $n'_i (1 \leq i \leq 6)$ be the number of exceptional vertices in σ^* whose configuration number is i , then $\text{CZ}(\sigma^*) = e^{-(n'_1 \epsilon_1 + n'_2 \epsilon_2 + \dots + n'_6 \epsilon_6) / k_B T}$. RCV(node S , state vector σ , orientation X) returns a converted vector σ^* . Let O be an orientation of $E(G[S])$ containing X and $cvec_O = \sigma$. Then $\sigma^* = cvec_{O, X}$.

4.2 Correctness and Complexity

This subsection describes correctness and time/memory complexity of Algorithm2.

Theorem 4. *Algorithm2 correctly evaluates the partition function of 4-regular graph of the ice-type model.*

Proof. For any node S , let $A_{S,\sigma}^*$ be the set of orientations of $E(G[S])$ whose state vector is σ . Let $H(S, O) = n_1\epsilon_1 + n_2\epsilon_2 + \dots + n_6\epsilon_6$ for any orientation O of $E(G[S])$ where n_i is the number of vertices whose $d_S \geq 3$ and whose configuration number is i on the orientation O . Let $Z(S, \sigma) = \sum_{O \in A_{S,\sigma}^*} e^{-(n_1\epsilon_1 + n_2\epsilon_2 + \dots + n_6\epsilon_6)/k_B T}$. The following argument shows that $Z(S, \sigma) = dp_S[\sigma]$ by induction.

When S is a leaf node, $\mathcal{S}_S = \{(0, 0, \dots, 0)\}$. Z and dp_S satisfy $Z(S, (0, 0, \dots, 0)) = dp_S[(0, 0, \dots, 0)] = 1$.

Next, we consider the case S is an internal node whose children L and R satisfy the induction hypothesis. Let X be an orientation of $E_{L,R}$ and $\sigma^* = \text{RCV}(S, \sigma, X)$. Let A_{X,S,σ^*}^{**} be a set of orientations of $E(G[S])$ such that its element O contains X and $\text{svec}_O = \sigma$. Note that for each vertex, directions of adjacent edges holding the restriction of σ^* are equal to those of both σ and X . Let $A_{X,S,\sigma_{S'}^*}^{part}(S') (S' \in \{L, R\})$ be a set of orientations of $E(G[S'])$ whose element O' satisfies $\text{cvec}_{O' \cup X, X} = \sigma_{S'}^*$. By Lemma 3, $A_{X,S,\sigma^*}^{**} = \{p \cup X \cup q \mid p \in A_{X,S,\sigma^* \circ e_L}^{part}(L), q \in A_{X,S,\sigma^* \circ e_R}^{part}(R)\}$ because the restriction of configuration is also dependent only on adjacencies of each vertex. Thus,

$$\begin{aligned} Z(S, \sigma) &= \sum_{O \in A_{S,\sigma}^*} e^{-H(S,O)/k_B T} \\ &= \sum_X \sum_{O \in A_{X,S,\sigma^*}^{**}} e^{-H(S,O)/k_B T} \\ &= \sum_X \left(\sum_{p \in A_{X,S,\sigma^* \circ e_L}^{part}(L)} e^{-H(L,p \cup X)/k_B T} \right) \left(\sum_{q \in A_{X,S,\sigma^* \circ e_R}^{part}(R)} e^{-H(R,X \cup q)/k_B T} \right) \\ &= \sum_X dp'_L[\sigma \circ e_L] dp'_R[\sigma \circ e_R] \end{aligned}$$

To get the last equation, it is enough to show the following lemma:

Lemma 5. *For any node S and its child $C \in \{L, R\}$, $\sum_{p \in A_{X,S,\sigma^* \circ e_C}^{part}(C)} e^{-H(C,p \cup X)/k_B T} = dp'_C[\sigma^* \circ e_C]$.*

Proof (Lemma 5). Let P be a set of a state vector σ of C satisfying $\sigma^* \circ e_C \in CV(C, \sigma, E_{L,R})$.

$$dp'_C[\sigma^* \circ e_C] = \sum_{\sigma \in P} CZ(\sigma^*) \cdot dp_C[\sigma] = \sum_{\sigma \in P} Z(C, \sigma) \cdot CZ(\sigma^*)$$

$Z(C, \sigma) \cdot CZ(\sigma^*)$ is equal to the sum of $e^{-H(C,p \cup X)/k_B T}$ for any orientation p of $E(G[C])$ whose state vector is σ . Therefore, $\sum_{\sigma \in P} Z(C, \sigma) \cdot CZ(\sigma^*) = \sum_{p \in A_{X,S,\sigma^* \circ e_C}^{part}(C)} e^{-H(C,p \cup X)/k_B T}$. \square

The above argument shows that Algorithm2 outputs $Z(G, (0, 0, \dots, 0))$, which is what we want to evaluate. \square

Theorem 5. *Algorithm2 has $O(k \cdot (2\sqrt{2})^k \cdot n)$ time complexity where k is the carving-width of the input graph.*

Proof. For each call of RecursivePartOfAlgorithm2, the total number of execution of lines in the loop of σ_C and σ_C^* is bounded by $O((2\sqrt{2})^k)$ because $|\mathcal{S}_S| \leq 2^k$ (Lemma 4) and $|\text{CV}| \leq \sqrt{2}^k$. By Lemma 4, that of the line dp_S is bounded by $2^{E_{S,V \setminus S} + E_{L,R}}$. By the same argument shown in Theorem 2, $2^{E_{S,V \setminus S} + E_{L,R}} \leq (2\sqrt{2})^k$. Since each function can be calculated in $O(k)$, the total time consumption for each recursion is $O(k \cdot (2\sqrt{2})^k)$. \square

Theorem 6. *Algorithm2 has $O(2^k \cdot n + (2\sqrt{2})^k)$ memory complexity where k is the carving-width of the input graph.*

Proof. The number of non-zero contents of dp'_C and dp_S is bounded by $|\mathcal{S}_C| \cdot |\text{CV}|, |\mathcal{S}_S|$ respectively. Lemma 4 shows that they are bounded by $O((2\sqrt{2})^k), O(2^k)$ respectively. We only need to store dp to calculate the dynamic programming table of its parents. \square

5 Conclusion

We introduced FPT algorithms based on a carving decomposition. When a carving-width is constant, as in ice problems, they are linear in time and space.

Our method focuses on the fact that for any subgraph induced by S , the number of degree vectors $|\mathcal{D}_S|$ is bounded by the number of edges between S and $V \setminus S$. So there is a possibility of finding an FPT algorithm of another parameter, especially if the parameter can limit the number of edges like carving-width.

Now we discuss another width parameter. For a clique-width, Fomin et al. [9] proved that the graph coloring problem is $W[1]$ -hard parameterized by clique-width, therefore we suppose that the problem of computing the Tutte polynomial is $W[1]$ -hard parameterized by clique-width. From the above, the existence of an FPT algorithm of a clique-width that counts the number of Eulerian orientations is unlikely.

Acknowledgment. The work by the third and fourth authors was supported in part by KAKENHI JP15H01677, JP16K12392, JP17K12639. The authors also thank anonymous reviewers for their helpful comments.

References

1. Andrzejak, A.: An algorithm for the Tutte polynomials of graphs of bounded treewidth. *Discret. Math.* **190**(1–3), 39–54 (1998)
2. Balaji, N., Datta, S., Ganesan, V.: Counting Euler tours in undirected bounded treewidth graphs. In: *Proceedings of the 35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science*, pp. 246–260 (2015)

3. Brightwell, G.R., Winkler, P.: Counting Eulerian circuits is $\#P$ -complete. In: Proceedings of the Seventh Workshop on Algorithm Engineering and Experiments and the Second Workshop on Analytic Algorithmics and Combinatorics, pp. 259–262 (2005)
4. Cai, J.-Y., Fu, Z., Shao, S.: A complexity trichotomy for the six-vertex model. [arXiv:1704.01657](https://arxiv.org/abs/1704.01657) [cs.CC] (2017)
5. Cai, J.-Y., Fu, Z., Xia, M.: Complexity classification of the six-vertex model. [arXiv:1702.02863](https://arxiv.org/abs/1702.02863) [cs.CC] (2017)
6. Chebolu, P., Cryan, M., Martin, R.: Exact counting of Euler tours for graphs of bounded treewidth. [arXiv:1310.0185](https://arxiv.org/abs/1310.0185) [cs.DM] (2013)
7. Creed, P.J.: Counting and sampling problems on Eulerian graphs. Ph.D. thesis, The University of Edinburgh (2010)
8. Euler, L.: *Solutio problematis ad geometriam situs pertinentis*. *Commentarii Academiae Scientiarum Petropolitanae* **8**, 128–140 (1741)
9. Fomin, F.V., Golovach, P.A., Lokshtanov, D., Saurabh, S.: Intractability of clique-width parameterizations. *SIAM J. Comput.* **39**(5), 1941–1956 (2010)
10. Ge, Q., Stefankovic, D.: The complexity of counting Eulerian tours in 4-regular graphs. *Algorithmica* **63**(3), 588–601 (2012)
11. Huang, S., Pinyan, L.: A dichotomy for real weighted Holant problems. *Comput. Complex.* **25**(1), 255–304 (2016)
12. Lieb, E.H.: Residual entropy of square ice. *Phys. Rev.* **162**(1), 162–172 (1967)
13. Mihail, M., Winkler, P.: On the number of Eulerian orientations of a graph. In: Proceedings of the Third Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, pp. 138–145 (1992)
14. Minc, H.: Upper bounds for permanents of $(0,1)$ -matrices. *Bull. Am. Math. Soc.* **69**, 789–791 (1963)
15. Noble, S.D.: Evaluating the tutte polynomial for graphs of bounded tree-width. *Comb. Probab. Comput.* **7**(3), 307–321 (1998)
16. Pauling, L.: The structure and entropy of ice and of other crystals with some randomness of atomic arrangement. *J. Am. Chem. Soc.* **57**(12), 2680–2684 (1935)
17. Sasák, R.: Comparing 17 graph parameters. Master’s thesis, The University of Bergen (2010). <http://bora.uib.no/handle/1956/4329>
18. Schrijver, A.: Bounds on the number of Eulerian orientations. *Combinatorica* **3**(3), 375–380 (1983)
19. Sekine, K., Imai, H., Tani, S.: Computing the Tutte polynomial of a graph of moderate size. In: Proceedings of the 6th International Symposium on Algorithms and Computation, pp. 224–233 (1995)
20. Seymour, P.D., Thomas, R.: Call routing and the Ratcatcher. *Combinatorica* **14**(2), 217–241 (1994)
21. Thilikos, D.M., Serna, M.J., Bodlaender, H.L.: Constructive linear time algorithms for small cutwidth and carving-width. In: Goos, G., Hartmanis, J., van Leeuwen, J., Lee, D.T., Teng, S.-H. (eds.) *ISAAC 2000*. LNCS, vol. 1969, pp. 192–203. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-40996-3_17
22. Valiant, L.G.: Holographic algorithms (Extended Abstract). In: Proceedings of the 45th Symposium on Foundations of Computer Science, pp. 306–315 (2004)
23. van der Waerden, B.L.: Aufgabe 45. *Jber. Deutsch. Math.-Verein.* **35**, 117 (1926)
24. Welsh, D.J.A.: The computational complexity of some classical problems from statistical physics. In: *Disorder in Physical Systems*, vol. 307, pp. 307–321. Oxford University Press (1990). <http://www.statslab.cam.ac.uk/~grg/books/jmh.html>
25. Welsh, D.J.A.: *Complexity: Knots, Colourings and Counting*. Cambridge University Press, Cambridge (1995)

On Structural Parameterizations of Happy Coloring, Empire Coloring and Boxicity

Jayesh Choudhari and I. Vinod Reddy^(✉)

IIT Gandhinagar, Gandhinagar, India
{choudhari.jayesh,reddy_vinod}@iitgn.ac.in

Abstract. Distance parameters are extensively used to design efficient algorithms for many hard graph problems. They measure how far a graph is from belonging to some class of graphs. If a problem is tractable on a class of graphs \mathcal{F} , then distances to \mathcal{F} provide interesting parameterizations to that problem. For example, the parameter vertex cover measures the closeness of a graph to an edgeless graph. Many hard problems are tractable on graphs of small vertex cover. However, the parameter vertex cover is very restrictive in the sense that the class of graphs with bounded vertex cover is small. This significantly limits its usefulness in practical applications. In general, it is desirable to find tractable results for parameters such that the class of graphs with the parameter bounded should be as large as possible. In this spirit, we consider the parameter distance to threshold graphs, which are graphs that are both split graphs and cographs. It is a natural choice of an intermediate parameter between vertex cover and clique-width. In this paper, we give parameterized algorithms for some hard graph problems parameterized by the distance to threshold graphs. We show that HAPPY COLORING and EMPIRE COLORING problems are fixed-parameter tractable when parameterized by the distance to threshold graphs. We also present an approximation algorithm to compute the BOXICITY of a graph parameterized by the distance to threshold graphs.

Keywords: Parameterized complexity · Threshold graphs
Algorithm

1 Introduction

Many interesting practical problems are NP-complete and it is unlikely that a polynomial time algorithm exists for one of these problems. To cope with hard problems many techniques have been introduced, one such technique is parameterized complexity theory. Unlike classical complexity theory, in parameterized complexity theory the running time of an algorithm is measured as a function of input size and an additional measure called the parameter. The aim here is to design algorithms that solve the problem in polynomial time in the size of the input such that the degree of the polynomial is independent of the parameter. These kinds of algorithms are called *fixed parameter tractable* (FPT) algorithms.

More precisely, we say that a problem is FPT if it can be solved in $f(d)n^{O(1)}$ time for some computable function f , where d is the value of the parameter and n is the size of the problem instance.

There may be several interesting parameterizations for any given problem. However, there are two popular approaches to select a parameter for optimization problems on graphs. First, the natural parameter is the size of solution (objective function). Second, the parameters which do not involve objective function, which are selected based on the structure of the graph called structural graph parameters. For example, the parameter vertex cover number measures the closeness of a graph to an independent set. In this paper we focus on some special group of structural parameters known as *distance-to-triviality* parameters [15]. They measure how far a graph is from some class of graphs where the problem is tractable. If a problem is known to be tractable on some class of graphs \mathcal{F} then the distance to \mathcal{F} gives an interesting parameterization to that problem. Our notion of distance to a graph class is the vertex deletion distance. For a class \mathcal{F} of graphs we say that $X \subseteq V(G)$ is an \mathcal{F} -modulator of a graph G if $G \setminus X \in \mathcal{F}$. If the size of the smallest modulator to \mathcal{F} is d , we say that the distance of G to the class \mathcal{F} is d .

Related Work. One of the well-studied parameter in the research of structural parameters is tree-width. The parameter tree-width measures closeness of graph from a tree (distance to “tree”). The notions of tree-decomposition and tree-width are commonly used to design efficient algorithms for hard problems. Courcelle [9] showed that every decision problem on graphs expressible in monadic second-order logic is fixed-parameter tractable when parameterized by tree-width of the input graph. Cai [8] studied the parameterized complexity of GRAPH COLORING with respect to several distance-to-triviality parameters. For instance, he showed that GRAPH COLORING is FPT when parameterized by edge deletion distance to split graphs, whereas, it is $W[1]$ -hard parameterized by vertex deletion distance to split graphs. It is para-NP-hard when parameterized by vertex or edge deletion distance to bipartite graphs. Bulian et al. [7] introduced the notion of elimination distance to triviality, which is a generalization of distance to triviality and showed that GRAPH ISOMORPHISM problem is FPT when parameterized by elimination distance to bounded degree. Hartung et al. [16] showed that 2-CLUB problem is FPT with respect to distance to cluster graphs and distance to co-cluster graphs. Das et al. [12] showed that the GRAPH MOTIF problem is FPT when parameterized by distance to threshold graphs. For a detailed survey of structural and distance parameters we refer the reader to [7, 18]

Our Contributions. The class of graphs we are interested in this paper are threshold graphs, which are graphs that are both split and cographs. The parameter we consider is distance to threshold graphs, which is a generalization of both vertex cover and distance to clique parameters. It is an intermediate parameter between vertex cover and clique-width (see Fig. 1). The problems we consider in this paper are FPT when parameterized by vertex cover. But the parameter

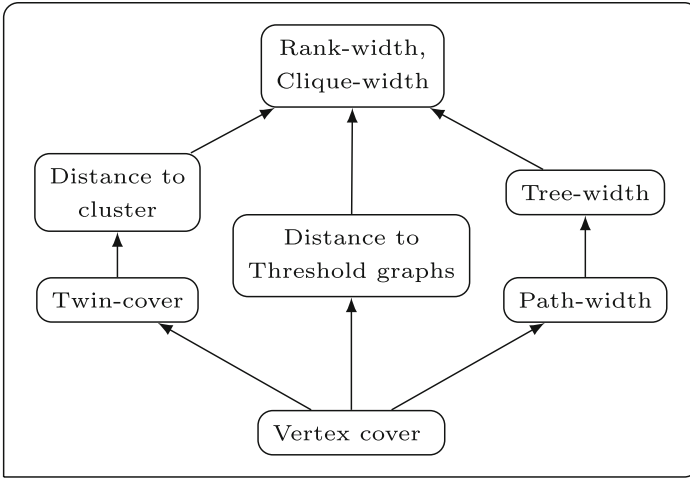


Fig. 1. A schematic showing the relation between the various parameters. An arrow from parameter *a* to *b* indicates that *a* is larger than *b*.

vertex cover is very restrictive and rarely useful for practical purposes. Studying the parameterized complexity of these problems with respect to distance to threshold graphs improves the understanding of tractable parameterizations.

The problems we study in this paper are HAPPY COLORING, EMPIRE COLORING and BOXICITY. We parameterize all these problems with distance to threshold graphs and obtain the following results:

- We give an FPT algorithm for HAPPY COLORING problem when parameterized by distance to threshold graphs.
- We present an FPT algorithm for EMPIRE COLORING problem when parameterized by distance to threshold graphs.
- We give $\left(2 + \frac{1}{\text{box}(G)}\right)$ -approximation algorithm for BOXICITY when parameterized by distance to threshold graphs, where $\text{box}(G)$ is the boxicity of the graph G .

2 Preliminaries

In this section, we introduce the notation and the terminology that we need to describe our algorithms. Most of our notation is standard. We use $[k]$ to denote the set $\{1, 2, \dots, k\}$. All graphs we consider in this paper are undirected, connected, finite and simple. For a graph $G = (V, E)$, let $V(G)$ and $E(G)$ denote the vertex set and edge set of G respectively. An edge in E between vertices x and y is denoted as xy for simplicity. For a subset $X \subseteq V(G)$, the graph $G[X]$ denotes the subgraph of G induced by vertices of X . Also, we abuse notation and use $G \setminus X$ to refer to the graph obtained from G after removing the vertex set X . $E[X, Y]$ denote the set of edges in G between X and Y . For a vertex $v \in V(G)$,

$N(v)$ denotes the set of vertices adjacent to v and $N[v] = N(v) \cup \{v\}$ is the closed neighborhood of v . A vertex is called *universal vertex* if it is adjacent to every other vertex of the graph.

A *cograph* is a graph which does not contain any P_4 (an induced path on four vertices). A *split graph* is a graph whose vertices can be partitioned into a clique and an independent. A graph is a *threshold graph* if it can be constructed from the one-vertex graph by repeatedly adding either an isolated vertex or a universal vertex (see Fig. 2). The class of threshold graphs is the intersection of split graphs and cographs [19]. We denote a threshold graph as $G = (C, I)$, where (C, I) denotes the partition of G into a clique and an independent set, respectively. It is easy to see that every induced subgraph of a threshold graph is also a threshold graph. We have the following characterization of threshold graphs:

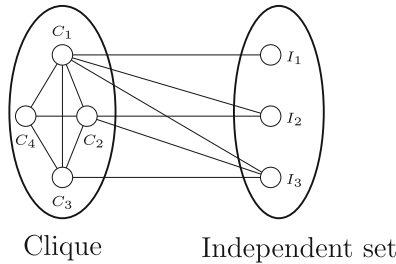


Fig. 2. Example of a threshold graph G , where each C_i and I_i represents a set of vertices having same neighbors. Every vertex of C_1 is a universal vertex in G .

Proposition 1 [19]. *For a graph $G = (V, E)$, the following statements are equivalent:*

1. G is a threshold graph.
2. G is a $(P_4, C_4, 2K_2)$ -free graph
3. For any $x, y \in V(G)$ either $N(x) \subseteq N[y]$ or $N(y) \subseteq N[x]$.

It is easy to see that checking whether a given graph G has vertex deletion distance d to the class of threshold graphs fixed-parameter tractable. The result follows from the fact that threshold graphs are characterized by a finite set of forbidden induced subgraphs (see Proposition 1). Therefore without loss of generality, in this paper, we assume that threshold graph modulator is given as a part of the input.

Parameterized Complexity. A parameterized problem denoted as $(I, d) \subseteq \Sigma^* \times \mathbb{N}$, where Σ is fixed alphabet and d is called the parameter. We say that the problem (I, d) is *fixed parameter tractable* with respect to parameter d if there exists an algorithm which solves the problem in time $f(d)|I|^{O(1)}$, where f is a computable function. For a detailed survey of the methods used in parameterized complexity, we refer the reader to the texts [11, 14].

3 Happy Coloring

Let G be an undirected vertex colored graph. A vertex v in G is *happy* if all the neighbors of v have color same as that of v . Given a partially colored graph G with ℓ colors, the MAX HAPPY VERTICES (ℓ -MHV) problem is to color all the remaining vertices such that the number of happy vertices is maximized. The ℓ -MHV problem is NP-hard [23] for $\ell \geq 3$ and solvable in polynomial time for $\ell \leq 2$. Recently in several papers [2-4,21] this problem has been studied from the parameterized complexity perspective. The problem is FPT when the parameter is either vertex cover number or distance to clique [21]. The MAX HAPPY VERTICES can be solved in polynomial time on cographs [21] using the notion of modular decomposition. In this section first we give a polynomial time algorithm for MAX HAPPY VERTICES problem on threshold graphs and this approach do not need the modular decomposition. Next, we present an FPT algorithm to this problem when parameterized by distance to threshold graphs.

MAX HAPPY VERTICES	Parameter: $d := X $
Input: A graph G , a partial coloring $p : S \rightarrow [\ell]$ for some $S \subseteq V(G)$, a positive integer k and a subset $X \subseteq V(G)$ such that $G \setminus X$ is a threshold graph	
Output: Is there a coloring $c : V(G) \rightarrow [\ell]$ such that $c _S = p$ and the number of happy vertices in G with respect to c is at least k ?	

Lemma 1. MAX HAPPY VERTICES problem can be solved in polynomial time on threshold graphs.

Proof. Let G be a threshold graph and $p : S \subseteq V(G) \rightarrow [\ell]$ is a partial coloring of G . Recall that every connected threshold graph contains a universal vertex. Therefore, the color of every happy vertex in any optimal coloring¹ is the same as the color of a universal vertex. If S contains two universal vertices u and v such that $p(u) \neq p(v)$ then no vertex of G is happy in any optimal coloring. If all universal vertices in S are colored with a single color by p , then coloring all uncolored vertices of G with that color maximizes the number of happy vertices. If S does not contain any universal vertices, then in any optimal coloring all universal vertices receive a single color, so we guess the color of the universal vertices in $O(\ell)$ time then color all the remaining uncolored vertices with the color of universal vertices. □

We now show that MAX HAPPY VERTICES is FPT parameterized by distance to threshold graphs.

Theorem 1. MAX HAPPY VERTICES problem is FPT when parameterized by the size of the modulator to threshold graphs.

Proof. Let (G, S, p, ℓ, k) be an instance of MAX HAPPY VERTICES problem. Let $X \subseteq V(G)$ of size d such that $G \setminus X = (C, I)$ is a threshold graph. We assume that

¹ A coloring is optimal if it maximizes the number of happy vertices.

both C and I are non-empty. If $C = \emptyset$ then X is a vertex cover of G and if $I = \emptyset$ then G is at most d distance to a clique. In both cases we use the algorithm given in [21] to solve the problem. We divide the proof into two cases based on the number of precolors (ℓ).

If $\ell \leq d$, the idea is to try over all possible colorings of X and for each coloring, we try to extend it to coloring of G . An optimal coloring is the one which maximizes the number of happy vertices. First, we guess the colors of uncolored vertices of X . We further guess the partition of X into X_h and X_u , where X_h and X_u denotes the set of happy and unhappy vertices in the coloring c . Note that if two vertices $x, y \in X_h$ have a common neighbor in $G \setminus X$, then the guessing X_h is invalid. For each vertex $x \in X_h$, color all its neighbors with the color of x . Now we need to color the uncolored vertices of $G \setminus X$. Let U be the set of universal vertices in the threshold graph $G \setminus X$. The main observation that we use here is that in any optimal coloring the color of every happy vertex in $G \setminus X$ is same as that of the universal vertices. If there exist two vertices $u, v \in U$ colored differently then no vertex of $G \setminus X$ is happy in any optimal coloring. In this case, we can arbitrarily color the remaining vertices of $G \setminus X$. If U has some colored vertices which are colored with the same color, then color all the remaining uncolored vertices of G with the color of u , where $u \in U \cap S$.

If no vertex of U is colored, then we guess the color of universal vertices. Since in any optimal coloring the color of every happy vertex in $G \setminus X$ is same as color of the universal vertices. So, we color all the remaining uncolored vertices of G with the color of universal vertices.

If $\ell > d$, then we partition the vertices of X into X_h and X_u . We further guess the partition $X_h = (X_{h_1}, \dots, X_{h_t})$ into $t \leq d$ partitions, where all vertices in $X_{h_i} \cup N(X_{h_i})$, $i \in [t]$ get the same color in an optimal coloring. Observe that for $i, j \in [t]$, $i \neq j$, $N(X_{h_i}) \cap N(X_{h_j}) = \emptyset$: if the intersection is non-empty then all the vertices in the intersection are not happy, contradiction to our guess.

Since X_{h_i} is happy there does not exist two vertices u and v in the set $X_{h_i} \cup N(X_{h_i})$ such that $p(u) \neq p(v)$. For each $i \in [t]$, if at least one vertex is precolored in $X_{h_i} \cup N(X_{h_i})$ then assign same color to all vertices of $X_{h_i} \cup N(X_{h_i})$. It remains to color the sets $X_{h_i} \cup N(X_{h_i})$, which do not have any precolored vertices. If $|S \cap U| \geq 2$ and if there exists u and v in $S \cap U$ such that $p(u) \neq p(v)$ then no vertex in $G \setminus X$ is happy. In this case, for each $i \in [t]$ we color all uncolored vertices of $X_{h_i} \cup N(X_{h_i})$ greedily with the color which did not appear in X_h so far. And if $|S \cap U| \leq 1$, then, first we guess the color of universal vertices U in $O(\ell)$ time. Next, we guess the set $X_{h_i} \cup N(X_{h_i})$ which gets the color of universal vertices. At the end we color all vertices of $X_{h_i} \cup N(X_{h_i})$ greedily with the color which did not appear in X_h so far.

Running time analysis. If $\ell \leq d$, then trying all possible colorings of X requires $O(d^d)$ time. Guessing the partition of X into X_h and X_u requires $O(2^d)$ time. Next, guessing the color of universal vertices takes $O(\ell)$ time. After guessing the color of X and universal vertices, we need $O(m+n)$ time to color the remaining vertices of the graph. So the whole algorithm takes $O(d^{d+1}2^d(m+n))$ time.

If $\ell > d$, then it takes $O(2^{d^d})$ time to guess the partition of X into X_h and X_u . Running through all possible ways of partitioning the vertices of $X_h = (X_{h_1}, \dots, X_{h_t})$ takes $O(d^d)$ time. Guessing the color of universal vertices takes $O(\ell)$ time, then guessing the set $X_{h_i} \cup N(X_{h_i})$ which gets the color of universal vertices takes time $O(t)$, and then to color the remaining vertices of the graph we need $O(m + n)$ time. So the algorithm runs in time $O(2^{d^d} d^d t \ell(m + n))$. \square

Li and Zhang [23] also studied an edge variant of the problem called MAXIMUM HAPPY EDGES (ℓ -MHE). It is defined as follows, in a vertex colored graph an edge is happy if both endpoints are colored with the same color. Given a partially vertex colored graph, the ℓ -MHE problem is to extend it to a total coloring of the graph such that the number of happy edges is maximized. Unfortunately, we could not resolve the parameterized complexity of this problem with respect to distance to threshold graphs. We do not even know whether this problem is polynomial solvable or NP-complete on threshold graphs. We leave them as open questions.

Open Problem 1. *Is ℓ -MHE FPT when parameterized by the distance to threshold graphs?*

Another important direction to pursue is to study the parameterized complexity of both ℓ -MHV and ℓ -MHE problems with respect to distance to cluster graphs. This is an intermediate parameter lies between vertex cover and clique-width [13]. It is not comparable with distance to threshold graphs (see Fig. 1).

Open Problem 2. *What is the parameterized complexity of ℓ -MHV and ℓ -MHE when parameterized by distance to cluster (cluster vertex deletion number)?*

4 Empire Coloring

Empire coloring problem was first studied by Heawood [17] in his 1890 paper in which he refuted a previous proof of the Four Color Theorem. The k -empire coloring problem is to color the vertices of G with at most k colors such that all vertices in a block get the same color and adjacent vertices belong to different blocks get distinct color. If the size of each block is one then the problem is equivalent to classical graph coloring problem. The problem is NP-hard on trees [20] for $k \geq 3$. We study the problem from the parameterized algorithmic perspective and show that the problem is FPT when parameterized by the distance to threshold graphs.

EMPIRE COLORING	Parameter: $d := X $
Input: A graph G , a partition B_1, \dots, B_t of $V(G)$, a positive integer k and a subset $X \subseteq V(G)$ such that $G \setminus X$ is a threshold graph	
Output: Is there an empire coloring of G using at most k colors?	

First, we show that the empire coloring problem can be reduced to a graph coloring problem on a smaller graph. Let $(G, (B_1, \dots, B_t), k)$ be an instance of

the empire coloring problem. We construct a new graph H from G as follows. The vertex set of H contains one vertex for each block, say $\{b_1, \dots, b_t\}$ and there is an edge between two vertices b_i and b_j in H if there exists two vertices $u \in B_i$ and $v \in B_j$ such that $uv \in E(G)$. The following Lemma is similar to the one used in [20].

Lemma 2. *Let G and H be the graphs as defined above. $(G, (B_1, \dots, B_t), k)$ has a k -empire coloring if and only if H has a k -coloring.*

Proof. Let C_G and C_H denote the k -empire coloring and k -coloring of graphs G and H respectively. Given a k -empire coloring C_G of G we construct a k -coloring of H as follows. If $C_G(B_i) = \{c\}$ then assign $C_H(b_i) = c$ for all $i \in [t]$. Suppose C_H is not a proper coloring of H , i.e., there exists two vertices $b_i, b_j \in V(H)$ such that $b_i b_j \in E(H)$ and $C_H(b_i) = C_H(b_j)$. This implies there exists $u \in B_i$ and $v \in B_j$ such that $uv \in E(G)$ and $C_G(u) = C_G(v)$ which is a contradiction to C_G is an empire coloring of G .

For the other direction, let C_H be a k -coloring of H . If $C_H(b_i) = c$, then for all $u \in B_i$ assign $C_G(u) = c$. It is easy to see that C_G is a k -empire coloring of G . \square

Corollary 1. *Let G and H be the graphs defined above. If G is a threshold graph, then H is also a threshold graph.*

Proof. The proof follows from Lemma 2 and the fact that every induced subgraph of a threshold graph is also a threshold graph. \square

Form the above corollary and using the fact that GRAPH COLORING problem is polynomial time solvable on threshold graphs, we can see that empire coloring problem on threshold graphs can be solved in polynomial time. We now show that empire coloring is FPT parameterized by the distance to threshold graphs.

Theorem 2. *EMPIRE COLORING problem is FPT when parameterized by size of the modulator to threshold graphs.*

Proof. Let $(G, (B_1^G, \dots, B_t^G), X, k)$ be an instance of the empire coloring problem, where $X \subseteq V(G)$ of size d such that $G \setminus X$ is a threshold graph. Define $D_i = B_i^G \cap (V(G) \setminus X)$ for all $i \in [t]$, then (D_1, \dots, D_t) is a partition of $G[V(G) \setminus X]$. Note that some D_i 's might be empty. We construct a graph G' from G by replacing all vertices in the set D_i with a vertex d_i , and there is an edge between two vertices d_i and d_j in $G' \setminus X$ if there exists two adjacent vertices $u \in D_i$ and $v \in D_j$. There is an edge between a vertex $x \in X$ and $d_i \in D_i$ if there exists a vertex $v \in D_i$ such that $xv \in E(G)$. For $x, y \in X$, $xy \in E(G')$ iff $xy \in E(G)$. Formally,

$$\begin{aligned}
 V(G') &= \{d_1, \dots, d_t\} \cup X \\
 E(G') &= E[X] \cup \{d_i d_j \mid \exists u \in D_i, v \in D_j, \text{ and } uv \in E(G)\} \\
 &\quad \cup \{x d_i \mid \exists x \in X, v \in D_i, \text{ and } xv \in E(G)\}
 \end{aligned}$$

Note that each vertex d_i in the independent set of the threshold graph $G' \setminus X$ is such that $B_i^G \cap C^G = \emptyset$, where C^G is the clique part of $V(G) \setminus X$. Also, the size of

the graph G' is at most $t + d$ (where, $t \leq k$) and $G' \setminus X$ is still a threshold graph (from Corollary 1). It is easy to see that $(G, (B_1^G, \dots, B_t^G), X, k)$ has a k -empire coloring if and only if $(G', (d_1 \cup (B_1^G \cap X), \dots, d_t \cup (B_t^G \cap X)), X, k)$ has a k -empire coloring.

Now, for each vertex d_i in the independent set of the threshold graph $G' \setminus X$, if $B_i^G \cap X \neq \emptyset$ then delete the vertex d_i and connect all the vertices in $N(d_i)$ to the vertices in $B_i^G \cap X$. Hereafter, the remaining vertices d_j in the independent set of $G' \setminus X$ are such that $B_j^G \cap X = \emptyset$ and $B_j^G \cap C^G = \emptyset$, where C^G is the clique part of $V(G) \setminus X$.

We partition the vertices of independent set I of G' based on their neighborhood in X . For a subset $U \subseteq X$ define $T_U^I := \{x \in I \mid N_{G'}(x) \cap X = U\}$ ($N_{G'}(x)$ is the neighborhood of x in G'). There are at most 2^d possible subsets in the partition of I (where $d = |X|$). For each $U \subseteq X$ there exists a vertex $v_U \in T_U^I$ such that $N_{G'}(w) \subseteq N_{G'}(v_U)$ for all $w \in T_U^I$, which implies all uncolored vertices in T_U^I can be colored with the color of v_U . This is because for any two vertices $v, w \in T_U^I$ the neighborhood of v and w is same in X , and as $G' \setminus X$ is a threshold graph, $\exists v_U \in T_U^I$ such that $N_{G'}(w) \subseteq N_{G'}(v_U)$.

Now, we build a new graph H from G' by replacing vertices in set T_U^I with vertex v_U for all $U \subseteq X$. The size of independent set I_H in H is at most 2^d . Observe that $H \setminus (X \cup I_H)$ is a clique with size of the modulator $(X \cup I_H)$ at most $d + 2^d$. So now we have an instance $(H, (B_1^H, \dots, B_t^H), k)$, where (B_1^H, \dots, B_t^H) is a partition of $V(H)$ and $H \setminus X$ is a clique. We solve the empire coloring on reduced graph H as follows.

We run through of all possible (at most $(d + 2^d)^{d+2^d}$) ways of precoloring $X \cup I_H$. For each precoloring of $X \cup I_H$, we test whether it is empire coloring or not. If the coloring is empire coloring then we extend it to color the clique $H \setminus X$ as follows. First for each clique vertex d_i in H if $B_i^H \cap X \neq \emptyset$ then color d_i with the color present in $B_i^H \cap X$. We color the remaining clique vertices by finding the maximum matching in following bipartite graph J .

The vertex set of the graph J contains uncolored clique vertices as one partition and precolors as the other partition. A vertex v is adjacent to a color c , if v is not adjacent to a vertex colored with c in H . Find the maximum matching in this bipartite graph J and we can color clique vertices based on this matching. We color the remaining clique vertices with new colors.

Running time analysis. The construction of graph H from G can be done in polynomial time. Trying all possible colorings of X takes $O((d + 2^d)^{d+2^d})$ time. For each possible coloring the bipartite graph can be build in $O(m)$ time and has at most n vertices. A maximum matching in a bipartite graph can be found in $O(n + \sqrt{nm})$ time. Therefore the running time of whole algorithm is $O((d + 2^d)^{d+2^d})(n + \sqrt{nm})(m + n)$. □

5 Boxicity

A k -box is a Cartesian product of k -intervals $[a_1, b_1] \times \dots \times [a_k, b_k]$. A k -box representation of a graph is a mapping of vertices to a k -box such that two vertices

are adjacent in G iff their corresponding k -boxes have non-empty intersection. The minimum value of k for which G has a k -box representation is called *boxicity* of G , denoted by $\text{box}(G)$. Alternatively, boxicity can be defined in terms of interval graphs [22]. The boxicity of a graph G is equal to the smallest integer k such that G can be expressed as the intersection of k interval graphs on the vertex set $V(G)$. Boxicity was introduced by Roberts [22] in 1969 and has applications in biology and ecology. Cozens [10] showed that computing boxicity of a graph is NP-complete. Several authors studied the problem from the parameterized complexity perspective. The problem is FPT when parameterized by the vertex cover number [1] and cluster vertex deletion number [6].

In this section, we give an upper bound for boxicity of a graph G in terms of the size of the threshold graph modulator. We also give a $(2 + \frac{1}{\text{box}(G)})$ -approximation algorithm for boxicity problem parameterized by size of the threshold graph modulator.

BOXICITY	Parameter: $d := X $
Input: A graph G , an integer k , a vertex subset $X \subseteq V(G)$ of size d such that $G \setminus X$ is a threshold graph.	
Output: Is $\text{box}(G) \leq k$?	

Lemma 3. *Let G be a graph and $X \subseteq V(G)$ of size d such that $G \setminus X = (C, I)$ is a threshold graph, then $\text{box}(G) \leq d + 1$.*

Proof. Let G_1 and G_2 be the graphs obtained from G , as follows.

$$V(G_1) = V(G), E(G_1) = E(X) \cup \{uv \mid u, v \in C \cup I\} \cup E[X, C \cup I]$$

$$V(G_2) = V(G), E(G_2) = \{uv \mid u, v \in X\} \cup E(C \cup I) \cup \{uv \mid u \in X, v \in C \cup I\}$$

It is easy to see that G is the intersection of G_1 and G_2 . The graph G_2 is a threshold graph with clique $C \cup X$ and independent set I . We know that every threshold graph is an interval graph [5], therefore boxicity of G_2 is one. Observe that $X \subseteq V(G_1)$ of size at most d such that $G_1 \setminus X$ is a clique, i.e., G_1 is at most d -distance to a clique. We now show that boxicity of the graph G_1 is at most d . Let $X = \{x_1, \dots, x_d\} \subseteq V(G_1)$ such that $G_1 \setminus X$ is a clique. Let H_1, \dots, H_d are the interval graphs defined as follows.

$$V(H_i) = V(G_1), E(H_i) = \{vx_i \mid v \in N_{G_1}(x_i)\} \cup \{uv \mid u, v \in V(G_1) \setminus \{x_i\}\}$$

It is easy to see that for each $i \in [d]$, H_i is an interval graph and $E(G_1) = \bigcap_{k=1}^d E(H_i)$. Therefore $\text{box}(G) \leq d + 1$, this concludes the proof of the lemma. \square

Theorem 3. *Let G be a graph and $X \subseteq V(G)$ of size d such that $G \setminus X$ is a threshold graph. Then we can find a t -boxicity of G such that $t \leq 2\text{box}(G) + 1$ in FPT-time.*

Proof. Let G be a graph and $X \subseteq V(G)$ such that $G \setminus X = (C, I)$. Let G_1 (resp. G_2) be the subgraph of G induced by vertices $X \cup C$ (resp. $X \cup I$). Observe that

$X \subseteq V(G_1)$ is a vertex cover of G_1 and $X \subseteq V(G_2)$ such that $G_2 \setminus X$ is a clique. We can compute the boxicity of the graphs G_1 and G_2 using the algorithms given in [1] and [6] respectively. Let G_3 be the threshold graph induced by vertices $C \cup I$. Since every threshold graph is an interval graph, the boxicity of G_3 is one. Now we define the graphs H_1 , H_2 and H_3 as follows.

$$V(H_1) = V(G_1) \cup I, E(H_1) = E(G_1) \cup \{uv \mid u \in V(G_1), v \in I\}$$

$$V(H_2) = V(G_2) \cup C, E(H_2) = E(G_2) \cup \{uv \mid u \in V(G_2), v \in C\}$$

$$V(H_3) = V(G_3) \cup X, E(H_3) = E(G_3) \cup \{uv \mid u \in V(G_3), v \in X\}$$

Note that adding universal vertices does not change the boxicity of the graph, therefore we have $\text{box}(G_i) = \text{box}(H_i)$ for $i \in \{1, 2, 3\}$. It is easy to see that the intersection of edge sets of graphs H_1 , H_2 and H_3 is $E(G)$. This implies $\text{box}(G) \leq \text{box}(G_1) + \text{box}(G_2) + 1$. Since $\text{box}(G_1), \text{box}(G_2) \leq \text{box}(G)$, We get a t -box representation of G consisting of at most $2\text{box}(G) + 1$ interval graphs. \square

Open Problem 3. *Is BOXICITY FPT when parameterized by the distance to threshold graphs?*

Acknowledgements. The authors are grateful to the anonymous referees for their valuable remarks and suggestions that significantly helped them improve the quality of the paper. The first author acknowledges support from Tata Consultancy Services (TCS) Research Fellowship.

References

1. Adiga, A., Chitnis, R., Saurabh, S.: Parameterized algorithms for boxicity. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010. LNCS, vol. 6506, pp. 366–377. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17517-6_33
2. Agrawal, A.: On the parameterized complexity of happy vertex coloring. In: International Workshop on Combinatorial Algorithms. Springer (2017, in press)
3. Aravind, N.R., Kalyanasundaram, S., Kare, A.S.: Linear time algorithms for happy vertex coloring problems for trees. In: Mäkinen, V., Puglisi, S.J., Salmela, L. (eds.) IWOCA 2016. LNCS, vol. 9843, pp. 281–292. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44543-4_22
4. Aravind, N., Kalyanasundaram, S., Kare, A.S., Lauri, J.: Algorithms and hardness results for happy coloring problems. arXiv preprint [arXiv:1705.08282](https://arxiv.org/abs/1705.08282) (2017)
5. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey. SIAM (1999)
6. Bruhn, H., Chopin, M., Joos, F., Schaudt, O.: Structural parameterizations for boxicity. *Algorithmica* **74**(4), 1453–1472 (2016)
7. Bulian, J.: Parameterized complexity of distances to sparse graph classes. Technical report, University of Cambridge, Computer Laboratory (2017)
8. Cai, L.: Parameterized complexity of vertex colouring. *Discrete Appl. Math.* **127**(3), 415–429 (2003)
9. Courcelle, B.: The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Inf. Comput.* **85**(1), 12–75 (1990)
10. Cozzens, M.B.: Higher and multi-dimensional analogues of interval graphs (1982)

11. Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer, Cham (2015). <https://doi.org/10.1007/978-3-319-21275-3>
12. Das, B., Enduri, M.K., Misra, N., Reddy, I.V.: On structural parameterizations of graph Motif and Chromatic number. In: Gaur, D., Narayanaswamy, N.S. (eds.) CALDAM 2017. LNCS, vol. 10156, pp. 118–129. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-53007-9_11
13. Doucha, M., Kratochvíl, J.: Cluster vertex deletion: a parameterization between vertex cover and clique-width. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 348–359. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32589-2_32
14. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity, vol. 4. Springer, London (2013). <https://doi.org/10.1007/978-1-4471-5559-1>
15. Guo, J., Hüffner, F., Niedermeier, R.: A structural view on parameterizing problems: distance from triviality. In: Downey, R., Fellows, M., Dehne, F. (eds.) IWPEC 2004. LNCS, vol. 3162, pp. 162–173. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28639-4_15
16. Hartung, S., Komusiewicz, C., Nichterlein, A., Suchý, O.: On structural parameterizations for the 2-club problem. *Discrete Appl. Math.* **185**, 79–92 (2015)
17. Heawood, P.J.: Map-colour theorem. *Proc. London Math. Soc.* **s2–51**(1), 161–175 (1949)
18. Lampis, M.: Structural parameterizations of hard graph problems. City University of New York (2011)
19. Mahadev, N.V., Peled, U.N.: Threshold Graphs and Related Topics, vol. 56. Elsevier, Amsterdam (1995)
20. McGrae, A.R., Zito, M.: The complexity of the empire colouring problem. *Algorithmica* **68**(2), 483–503 (2014)
21. Misra, N., Reddy, I.V.: The parameterized complexity of happy colorings. arXiv preprint [arXiv:1708.03853](https://arxiv.org/abs/1708.03853) (2017)
22. Roberts, S.: On the boxicity and cubicity of a graph. In: *Recent Progresses in Combinatorics*, pp. 301–310 (1969)
23. Zhang, P., Li, A.: Algorithmic aspects of homophily of networks. *Theor. Comput. Sci.* **593**, 117–131 (2015)

Complexity of the Maximum k -Path Vertex Cover Problem

Eiji Miyano¹, Toshiki Saitoh¹, Ryuhei Uehara², Tsuyoshi Yagita^{1(✉)},
and Tom C. van der Zanden³

¹ Kyushu Institute of Technology, Iizuka, Japan

{miyano,toshikis}@ces.kyutech.ac.jp, yagita@theory.ces.kyutech.ac.jp

² Japan Advanced Institute of Science and Technology, Nomi, Japan
uehara@jaist.ac.jp

³ Utrecht University, Utrecht, The Netherlands
T.C.vanderZanden@uu.nl

Abstract. This paper introduces the maximum version of the k -path vertex cover problem, called the MAXIMUM k -PATH VERTEX COVER problem ($\text{Max}P_k\text{VC}$ for short): A path consisting of k vertices, i.e., a path of length $k - 1$ is called a k -path. If a k -path P_k includes a vertex v in a vertex set S , then we say that S or v covers P_k . Given a graph $G = (V, E)$ and an integer s , the goal of $\text{Max}P_k\text{VC}$ is to find a vertex subset $S \subseteq V$ of at most s vertices such that the number of k -paths covered by S is maximized. $\text{Max}P_k\text{VC}$ is generally NP-hard. In this paper we consider the tractability/intractability of $\text{Max}P_k\text{VC}$ on subclasses of graphs: We prove that $\text{Max}P_3\text{VC}$ and $\text{Max}P_4\text{VC}$ remain NP-hard even for split graphs and for chordal graphs, respectively. Furthermore, if the input graph is restricted to graphs with constant bounded treewidth, then $\text{Max}P_3\text{VC}$ can be solved in polynomial time.

1 Introduction

One of the most important and most fundamental computational problems in graph theory, combinatorial optimization, and theoretical computer science is the MINIMUM VERTEX COVER problem (MinVC). Indeed, as one of the seminal results in computational complexity theory, the decision version of MinVC was listed in Karp's original 21 NP-complete problems in [10].

Very recently, Brešar, Kardoš, Katrenič, and Semanišin introduced a generalized variant of MinVC , called the MINIMUM k -PATH VERTEX COVER problem ($\text{Min}P_k\text{VC}$), motivated by the need to secure the data integrity of wireless sensor networks from attackers [5]: Let $G = (V, E)$ be a simple undirected graph, where V and E denote the set of vertices and the set of edges, respectively. $V(G)$ and $E(G)$ also denote the vertex set and the edge set of G , respectively. A path consisting of k vertices, i.e., a path of length $k - 1$ is called a k -path. If a k -path

This work is partially supported by JSPS KAKENHI Grant Numbers JP16K16006, JP17H06287, JP17K00016, JP24106004, JP26330009, and JST CREST JPMJCR1402.

P_k contains a vertex v in a vertex set S , then we say that the set S or the vertex v covers P_k . Given a graph G , the goal of $\text{Min}P_k\text{VC}$ is to find a vertex subset $S \subseteq V(G)$ of *minimum* cardinality such that S covers all the k -paths in G . In the same paper, Brešar et al. proved the NP-hardness of $\text{Min}P_k\text{VC}$ and designed a linear-time algorithm for $\text{Min}P_k\text{VC}$ on trees for $k \geq 3$. Furthermore, the authors proved that $\text{Min}P_k\text{VC}$ can be expressed by Extended Monadic Second Order Logic, which implies that $\text{Min}P_k\text{VC}$ can be solved in linear time on graphs with bounded treewidth by Courcelle's theorem [8]. Subsequently, due to its wide applicability to many practical problems, $\text{Min}P_k\text{VC}$ has been studied intensively. Indeed, for example, a large number of results on approximation [6, 12, 15, 16, 19], fixed-parameter tractability [11, 14] and exact algorithms [17] for $\text{Min}P_3\text{VC}$ and $\text{Min}P_4\text{VC}$ have been reported.

The classical/original MinVC has several variants; one of the most popular variants is the MAXIMUM VERTEX COVER problem (MaxVC), which is often called the PARTIAL VERTEX COVER problem: Given a graph G and an integer s , the goal of MaxVC is to find a vertex subset $S \subseteq V(G)$ of s vertices such that the number of edges covered by S is *maximized*. It is known that MaxVC also has many applications in real life (see, e.g., [7]). It is known [1, 7] that MaxVC is NP-hard even on bipartite graphs, though the minimization version MinVC is solvable in polynomial time on them.

For the general version $\text{Min}P_k\text{VC}$, therefore, it would be natural to consider the maximization problem; this paper introduces the MAXIMUM k -PATH COVER problem ($\text{Max}P_k\text{VC}$): Given a graph G and an integer s , the goal of $\text{Max}P_k\text{VC}$ is to find a vertex subset $S \subseteq V(G)$ of size at most s such that the number of k -paths covered by S is *maximized*. One can see that $\text{Max}P_2\text{VC}$ is generally NP-hard since it is identical to MaxVC . Therefore, we focus on the case where $k \geq 3$. For any fixed integer $k \geq 3$, $\text{Max}P_k\text{VC}$ is NP-hard in the general case since $\text{Min}P_k\text{VC}$ can be considered as a special case of $\text{Max}P_k\text{VC}$. In this paper, we are interested in the tractability and the intractability of $\text{Max}P_k\text{VC}$ on subclasses of graphs.

Our main results are summarized as follows:

- (i) $\text{Max}P_3\text{VC}$ remains NP-hard for the class of split graphs.
- (ii) $\text{Max}P_4\text{VC}$ remains NP-hard for the class of chordal graphs.
- (iii) $\text{Max}P_3\text{VC}$ can be solved in polynomial time if the input graph is restricted to graphs with constant bounded treewidth.

Due to the page limitation, we omit some proofs from this extended abstract.

2 Preliminaries

Let $G = (V, E)$ be a simple undirected graph, where V and E denote the set of vertices and the set of edges, respectively. $V(G)$ and $E(G)$ also denote the vertex set and the edge set of G , respectively. We denote an edge with endpoints u and v by $\{u, v\}$. A path of length $k - 1$ from a vertex v_1 to a vertex v_k is represented as a sequence of vertices such that $P_k = \langle v_1, v_2, \dots, v_k \rangle$, which is called a k -path.

For a vertex v , the set of vertices adjacent to v , i.e., the *open neighborhood* of v is denoted by $N(v)$. Let $\text{deg}(v) = |N(v)|$ be the *degree* of v . Let $G[S]$ denote the subgraph of G induced by a vertex subset $S \subseteq V(G)$.

A graph G is *chordal* if each cycle in G of length at least four has at least one chord, where the chord of a cycle is an edge between two vertices of the cycle that is not an edge of the cycle. A graph G is *split* if there is a partition of $V(G)$ into a clique set V_1 and an independent set V_2 such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V(G)$. A *treewidth* of a graph is defined in Sect. 5.

The problem $\text{Max}P_k\text{VC}$ that we study in this paper is defined as follows for any fixed integer k :

MAXIMUM k -PATH VERTEX COVER ($\text{Max}P_k\text{VC}$)

Given a graph G and an integer s , the goal of $\text{Max}P_k\text{VC}$ is to find a vertex subset $S \subseteq V(G)$ of size at most s such that the number of k -paths covered by S is maximized.

As mentioned in Sect. 1, it is known [5] that the minimum variant $\text{Min}P_k\text{VC}$ of our problem is NP-hard for any fixed integer $k \geq 2$. It is important here to note that $\text{Min}P_k\text{VC}$ can be considered as a special case of $\text{Max}P_k\text{VC}$, i.e., the essentially equivalent goal of $\text{Min}P_k\text{VC}$ is to find a vertex subset S of size at most s such that S covers all the k -paths in the input graph. Therefore, the NP-hardness of $\text{Max}P_k\text{VC}$ is straightforward:

Theorem 1. [5] *For any fixed integer $k \geq 2$, $\text{Max}P_k\text{VC}$ is NP-hard.*

Moreover, it is known that $\text{Min}P_3\text{VC}$ is a *dual* problem of the MAXIMUM DISSOCIATION SET problem, which was introduced in [18]. Yannakakis [18], and Papadimitriou and Yannakakis [13] proved that the problem is NP-hard even on bipartite graphs, and on planar graphs, respectively. Similarly to the above, we can obtain the following theorem:

Theorem 2. [13, 18] *$\text{Max}P_3\text{VC}$ is NP-hard on (i) bipartite graphs and (ii) planar graphs.*

3 NP-Hardness of $\text{Max}P_3\text{VC}$ on Split Graphs and $\text{Max}P_4\text{VC}$ on Chordal Graphs

In this section, we prove the NP-hardness of $\text{Max}P_3\text{VC}$ on split graphs and $\text{Max}P_4\text{VC}$ on chordal graphs. Let us define a decision version of $\text{Max}P_3\text{VC}$, denoted by $\text{Max}P_3\text{VC}(t)$: Given a graph G , and two integers s and t , determine if the graph G has a vertex subset $S \subseteq V(G)$ of size at most s such that the total number of 3-paths covered by S is at least t . The first result of this section is:

Theorem 3. *$\text{Max}P_3\text{VC}(t)$ is NP-complete, even on split graphs.*

Proof. First, we prove that $\text{Max}P_3\text{VC}(t)$ is in NP. Every path of three vertices in the graph G can be enumerated in $O(|V|^3)$ time, thus if we nondeterministically guess a set S of s vertices, we can check whether at least t 3-paths are covered by those s vertices in polynomial time.

Next, we show that there exists a polynomial-time reduction from the RESTRICTED EXACT COVER BY THREE SETS (RX3C) problem to $\text{Max}P_3\text{VC}(t)$. The input is a finite set $X = \{x_1, x_2, \dots, x_{3q}\}$ of $3q$ elements and a collection \mathcal{C} of $3q$ 3-element subsets of X , where each element of X appears in exactly three subsets of \mathcal{C} . RX3C asks if \mathcal{C} contains an exact cover for X , that is, a subcollection $\mathcal{C}' \subseteq \mathcal{C}$ such that every element of X occurs in exactly one member of \mathcal{C}' . RX3C is shown to be NP-complete by Gonzalez [9]. We give the reduction such that the original instance of RX3C is a yes-instance if and only if the $\text{Max}P_3\text{VC}(t)$ instance is also a yes-instance. Let $n = 3q$ for a while. As an input of RX3C, let $X = \{x_1, x_2, \dots, x_n\}$ be a set of n elements. Also, let $\mathcal{C} = \{C_1, C_2, \dots, C_n\}$ be a collection of n 3-element sets. Then, we construct a graph $G = (V, E)$ corresponding to an instance (X, \mathcal{C}) of RX3C as follows: The constructed graph G consists of the following vertices: (i) n vertices, v_{C_1} through v_{C_n} , called the *set vertices*, corresponding to the n sets, C_1 through C_n , respectively, (ii) n vertices, v_{x_1} through v_{x_n} , called the *element vertices*, corresponding to the n elements, x_1 through x_n , respectively, and (iii) corresponding to each set C_i ($i \in \{1, 2, \dots, n\}$), n^2 vertices, $v_{C_i,1}$ through v_{C_i,n^2} , i.e., n^3 vertices in total, called *pendant vertices*. Let $C = \{v_{C_1}, v_{C_2}, \dots, v_{C_n}\}$, $EL = \{v_{x_1}, v_{x_2}, \dots, v_{x_n}\}$, and $P = \{v_{C_1,1}, \dots, v_{C_1,n^2}, v_{C_2,1}, \dots, v_{C_n,n^2}\}$. The edge set $E(G)$ is as follows: (iv) The subgraph induced by the set C of n vertices forms a clique K_n of size n , i.e., we add all possible edges between any pair of vertices in C into $E(G)$. (v) If $x_i \in C_j$ for $i \in \{1, 2, \dots, n\}$ and $j \in \{1, 2, \dots, n\}$, then we add an edge $\{v_{x_i}, v_{C_j}\}$ into $E(G)$. Note that each set vertex v_{C_i} is adjacent to exactly three element vertices and furthermore each element vertex v_{x_j} is adjacent to exactly three set vertices. (vi) For each $i \in \{1, 2, \dots, n\}$ and each $j \in \{1, 2, \dots, n^2\}$, the pendant vertex $v_{C_i,j}$ is connected to v_{C_i} by adding an edge $\{v_{C_i,j}, v_{C_i}\}$. Finally, we set $s = q$ and $t = 81q^5/2 + 45q^4 + 23q^3 + 15q^2/2 + 7q$. This completes the reduction, which clearly can be done in polynomial time. One can verify that the constructed graph G is split since the set vertices form a clique, and the remaining vertices form an independent set.

As an example, if we are given $X = \{1, 2, 3, 4, 5, 6\}$ and a collection $\mathcal{C} = \{C_1, C_2, \dots, C_6\} = \{\{1, 3, 5\}, \{1, 4, 5\}, \{3, 4, 6\}, \{2, 4, 6\}, \{1, 2, 6\}, \{2, 3, 5\}\}$ as an RX3C instance, the graph constructed above is illustrated in Fig. 1. One can see that $\mathcal{C}' = \{C_1, C_4\}$ is a possible solution.

Before showing the correctness of our reduction, we make important observations: (1) Each set vertex v_{C_i} can cover at least $\binom{n^2}{2} = \Omega(n^4)$ 3-paths, i.e., $\langle v_{C_i,j}, v_{C_i}, v_{C_i,k} \rangle$ for $1 \leq j, k \leq n^2$ and $j \neq k$. (2) On the other hand, every element or pendant vertex can cover at most $O(n^2)$ 3-paths. Therefore, in order to cover as many 3-paths as possible, it would be the most effective to select set vertices into a solution of $\text{Max}P_3\text{VC}(t)$.

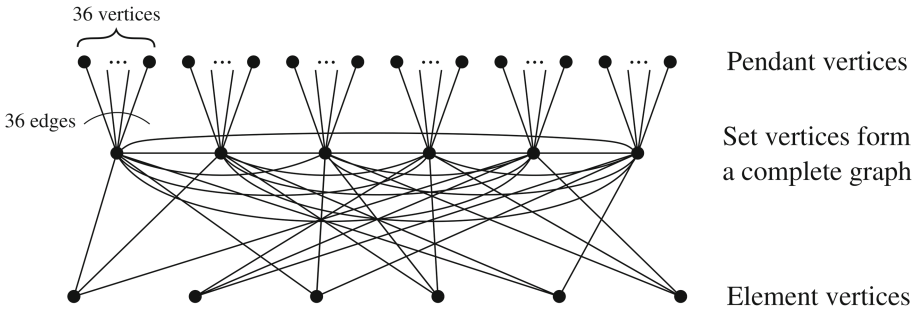


Fig. 1. Constructed graph G

The following lemma shows the correctness of the reduction:

Lemma 1. *RX3C is yes if and only if $\text{MaxP}_3\text{VC}(t)$ is yes, i.e., there is a vertex subset S of size at most q such that S can cover at least $81q^5/2 + 45q^4 + 23q^3 + 15q^2/2 + 7q$ 3-paths.*

This completes the proof of Theorem 3. □

By using a very similar reduction with small modification, we can obtain the following theorem:

Theorem 4. *$\text{MaxP}_4\text{VC}(t)$ is NP-complete, even on chordal graphs.*

4 Algorithm for MaxP_3VC on Trees

In the next section we present a polynomial-time algorithm for MaxP_3VC on graphs with bounded treewidth, but, in order to make our basic ideas clear, this section provides a simpler algorithm running in polynomial time only for MaxP_3VC on trees. In the following, let T denote the given tree, and especially, let $T_{v_{root}}$ denote the subtree of T whose root is v_{root} .

Intuitively, our algorithm is based on dynamic-programming, keeping the *minimum number of uncovered 3-paths* from the bottom to the top of the tree. For every vertex, the following two steps are considered in our algorithm: [I] **Introduce Step** and [II] **Join Step**, and in each step, the table in which the minimum number of uncovered 3-paths is stored is updated. After computing the minimum number of uncovered 3-paths of a certain subtree, our algorithm proceeds to the parent vertex u of the root of the subtree. Then, we say that u is in Introduce Step (see Fig. 2). Also, there may exist some subtrees whose parent of the root of each subtree is u . In such case, our algorithm merges those subtrees one by one, by joining the same parent u , and computes the minimum number of uncovered 3-paths. In this joining step, we say u is in Join Step (see Fig. 3).

Now, we are going to show the recursive formulas with precise notation. Let $c[v; b, \ell, r]$ denote the number of uncovered 3-paths, where v denotes the vertex

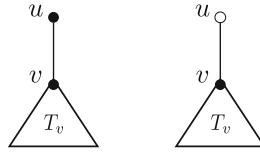


Fig. 2. Vertex u is in Introduce Step; u is in the cover (Left), or not (Right)

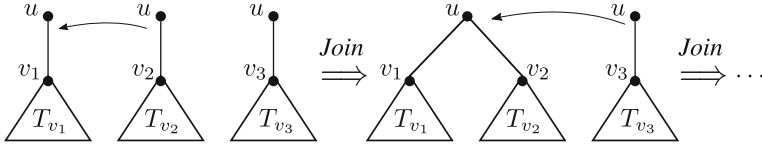


Fig. 3. Vertex u is in Join Step

Table 1. Table when the root is in the cover

Solution size	-
0	∞
1	<i>in_1</i>
2	<i>in_2</i>
\vdots	\vdots
s	<i>in_s</i>

we are currently looking at, $b \in \{1, 0\}$ denotes whether the vertex is selected in the cover ($b = 1$) or not ($b = 0$) as a root, $\ell \in \{0, 1, 2, \dots, s\}$ denotes the size of the solution, and $r \in \{0, \dots, n - 1\}$ denotes the number of unselected children. Note that, when a vertex is in Introduce Step and chosen in the cover, we do not need to consider the fourth argument r . This is because the 3-paths including the vertex in Introduce Step and its children are already covered by the vertex in Introduce Step. We show the tables in Tables 1 and 2, where each entry denotes the minimum number of uncovered 3-paths under a set of some arguments. When a vertex v is in Introduce Step, first we consider two cases; $b = 1$ or $b = 0$, that is, whether we put v in the cover or not. If $b = 1$, then we only consider the solution size ℓ , ranging it from 1 to s . For example, in Table 1, *in_1* stands for $c[v; 1, 1, *]$, and there is stored the minimum number of uncovered 3-paths under these conditions. Similarly, if $b = 0$, then there are $s + 1$ and n options for the solution size and the number of unselected children of the root. Each of the entry, such as *out_00*, *out_01* and so on, stores the minimum number of the 3-paths with each argument. Utilizing this table, the algorithm proceeds from the bottom to the top.

Leaf: If the vertex u is a leaf, then there are no uncovered 3-paths, thus we have $c[u; 0, 0, -] = c[u; 1, 1, -] = 0$.

Table 2. Table when the root is NOT in the cover

Solution size	The number of children of the root NOT chosen in cover			
	0	1	...	$n - 1$
0	<i>out_00</i>	<i>out_01</i>	...	<i>out_0</i> ($n - 1$)
1	<i>out_10</i>	<i>out_11</i>	...	<i>out_1</i> ($n - 1$)
2	<i>out_20</i>	<i>out_21</i>		<i>out_2</i> ($n - 1$)
⋮	⋮	⋮	⋮	⋮
s	<i>out_s0</i>	<i>out_s1</i>		<i>out_s</i> ($n - 1$)

Introduce Step: If the vertex u is in Introduce Step, assuming v , the child vertex of u , has d children, we consider two cases: u is in the cover or not.

(i) **u is in the cover:** As mentioned before, we do not need to consider the fourth argument, so we have only to take care of the size of the solution which ranges from 1 to s . If the size of the solution is 1, then we refer to *out_d* of v , $c[v; 0, 0, d]$. This is because the root v is in the cover and the solution size we assume now is 1, v is not in the cover and the size of the solution for v is 0, and also v has d unselected children. If the solution size is 2, then it becomes little complicated. We have to take the minimum of $\{in_1, out_1d, out_1(d - 1)\}$ of v . This is because if the solution size is 2, from the assumption that we put u into the cover set, then we have to consider where one more vertex in the cover is in the subtree T_u . There are following three options in this case: (i) v is also in the cover set, (ii) even the children of v do not have the selected vertex, in other words, all of the d children of v are all unselected vertices, and (iii) one of the d children is in the cover set. Thus we refer three entries, and take the minimum of them. In the same manner, $c[u; 1, *, -]$ is calculated as follows, and also the table for u when the root is in (see Table 1) is updated with the following values:

$$c[u; 1, i, -] = \begin{cases} \infty & \text{if } i = 0 \\ c[v; 0, 0, d] & \text{if } i = 1 \\ \min_{0 \leq j \leq i-1} \{c[v; 1, i - 1, *], c[v; 0, i - 1, d - j]\} & \text{if } 2 \leq i \leq s \end{cases}$$

(ii) **u is not in the cover:** Since G is tree, we do not have to consider the case where the number of unselected children is $2, \dots, n - 1$. Thus we can set all the entries of Table 2 whose number of unselected children is $2, \dots, n - 1$ with ∞ . In other words, we have only to consider the case where the number of unselected children is 0 or 1. Furthermore, if u has 0 unselected children (which means v is in the cover) and the solution size is $1, \dots, s$, it is clear that we refer to the root-in table of v , corresponding to the solution size. If u has 1 uncovered child (which similarly means v is not in the cover), then we

have to take the minimum depending on the solution size. Thus $c[u; 0, *, *]$ is calculated as follows:

$$c[u; 0, i, j] = \begin{cases} \infty & \text{if } i = 0 \text{ and } j = 0 \\ c[v; 1, i, -] & \text{if } 1 \leq i \leq s \text{ and } j = 0 \\ c[v; 0, 0, d] & \text{if } i = 0 \text{ and } j = 1 \\ \min_{0 \leq d' \leq i} \{c[v; 0, i, d - d'] + d - d'\} & \text{if } 1 \leq i \leq s \text{ and } j = 1 \end{cases}$$

Join Step: If the vertex u is in Join Step, then we update the table of u . As with **Introduce Step**, we consider two cases: the root is in the cover or not. Let us assume that u is in Join Step, and let v_L and v_R be the left and right child of u , respectively. Also, for clarity, we specially introduce u_L and u_R such that $u = u_L = u_R$, whose child is v_L and v_R respectively.

(i) **u is in the cover:** We do not have to consider the fourth argument, as we mentioned. We update the table ranging the size of the solution from 0 to s . If the size of the solution is 0, we set the entry as ∞ . If the size of the solution is 1, then we just add the number of uncovered 3-paths $c[u_L; *, *, *]$ and $c[u_R; *, *, *]$. Note that u is selected in the cover, therefore the solution S has only u in this case. If the size of the solution is 2, then we have to take the minimum from two choices: one more solution is in the left subtree or the right subtree, say T_{v_L} or T_{v_R} . Thus $c[u; 1, *, -]$ is calculated as follows:

$$c[u; 1, i, -] = \begin{cases} \infty & \text{if } i = 0 \\ \min_{1 \leq j \leq i} \{c[u_L; 1, i - j + 1, -] + c[u_R; 1, j, -]\} & \text{if } 1 \leq i \leq s \end{cases}$$

(ii) **u is not in the cover:** If the vertex which is not selected in the cover is in Join Step, then uncovered 3-paths whose central vertex is u , in other words, the uncovered 3-paths going through from the left subtree T_{u_L} to the right subtree T_{u_R} may exist. Thus we have to take them into consideration in updating the table of u . There are two tables for subtrees T_{u_L} and T_{u_R} , so we have to take the minimum among all the possible combinations of the size of the solution and the number of unselected children of those subtrees, considering newly appearing uncovered 3-paths going from T_{u_L} to T_{u_R} . Note that these newly appearing uncovered 3-paths can be calculated by multiplying the two numbers of unselected children, the number of the unselected children in T_{u_L} and T_{u_R} . Let i, i_L , and i_R be the variables which respectively denotes the size of the solution in the subtree T_u, T_{u_L} , and T_{u_R} . Note that $i_R = i - i_L$ holds. Also, let j, j_L , and j_R be the variables which respectively denotes the number of the unselected children in the subtree T_u, T_{u_L} , and T_{u_R} . Note that $j_R = j - j_L$ also holds. $c[u; 0, *, *]$ is calculated as follows:

$$c[u; 0, i, j] = \min_{0 \leq i \leq s} \min_{0 \leq j \leq n-1} \{c[u_L; 0, i_L, j_L] + c[u_R; 0, i_R, j_R] + j_L \cdot j_R\}$$

Note that, since we can assume that for any vertex v , the number of unselected children r is always at least $deg(v) - s - 1$, the number of cases in the dynamic programming table is $O(s^2)$. The running time for the algorithm is dominated for that of **Join Step**, which (using this observation) is $O(s^4)$.

Theorem 5. *MaxP₃VC on trees of n vertices can be solved in $O(s^4 \cdot n)$ time, where s is the prescribed size of the 3-path vertex cover.*

5 Algorithm for MaxP₃VC on Graphs with Bounded Treewidth

In this section, we show that MaxP₃VC admits a polynomial-time algorithm for graphs with bounded treewidth. In particular, we show that there exists an $O((s + 1)^{2tw+4} \cdot 4^{tw} \cdot n)$ -time algorithm, where tw denotes the *treewidth*, which is defined later. Thus, MaxP₃VC is in XP with respect to the parameter treewidth (and FPT with respect to the combined parameter $s + tw$).

Our algorithm uses dynamic programming on a *nice tree decomposition* [2] of the input graph G . Given a graph G , a *tree decomposition* of G is a tree T with for each node $v_T \in V(T)$ a subset $X_{v_T} \subseteq V(G)$ (called *bag*) such that

- for every $(u, v) \in E(G)$, there is a $v_T \in V(T)$ such that $\{u, v\} \subseteq X_{v_T}$, and
- for every $v \in V(G)$, the subset $\{v_T \in V(T) \mid v \in X_{v_T}\}$ induces a connected subtree of T .

The *width* of a tree decomposition is $\max_{v_T \in T} |X_{v_T}| - 1$, and the *treewidth* of a graph G is the minimum width taken over all tree decompositions of G . To avoid confusion, from now on we shall refer to the vertices of T as “nodes”, and “vertex” shall refer exclusively to vertices of G .

We designate an arbitrary node of T as *root* of the tree decomposition. Given a node $v_T \in T$, we denote by $G[v_T]$ the subgraph of G induced by X_{v_T} and the vertices in bags of nodes which are descendants of v_T in T . We moreover assume that our (rooted) decomposition is *nice*, that is, each of the nodes $v_T \in T$ is one of the four types:

- **Leaf:** v_T is a leaf of T , and $|X_{v_T}| = 1$.
- **Introduce:** v_T has a single child node u_T , and X_{v_T} differs from X_{u_T} only by the inclusion of one additional vertex w . We say that w is *introduced* in v_T .
- **Forget:** v_T has a single child node u_T . X_{v_T} differs from X_{u_T} only by the removal of one vertex w . We say that vertex w is *forgotten* in v_T .
- **Join:** v_T has two children u_T and u'_T . Moreover, $X_{u_T} = X_{u'_T} = X_{v_T}$.

We note that a tree decomposition can be converted into a nice tree decomposition of the same width. Moreover, we can assume that the size of a tree decomposition (i.e. the number of bags) is linear in $|V(G)|$ [2].

Given a node v_T of a tree decomposition of G , a *partial solution* is a subset $S \subseteq V(G[v_T])$ of size at most s . Since the number of 3-paths in G is equal to

$$\sum_{v \in V} \frac{1}{2} \deg(v)(\deg(v) - 1),$$

we define the cost of a partial solution (relative to a node v_T) S to be

$$\sum_{v \in V(G[v_T]) \setminus (S \cup X_{v_T})} \frac{1}{2} \deg_{v_T, S}(v)(\deg_{v_T, S}(v) - 1),$$

where $\deg_{v_T, S}(v)$ is taken to be the degree of v in the subgraph of G induced by $V(G[v_T]) \setminus S$. This definition, which does not take into account the degrees of the vertices in X_{v_T} , is convenient because the degrees of the vertices in X_{v_T} are not yet fixed, and may change as new vertices are introduced. However, if we assume the root bag of the tree decomposition is empty (which may be accomplished by introducing a series of forget bags after the root bag), then a partial solution with minimum cost corresponds to an optimal solution to the $\text{Max}P_3\text{VC}$ instance.

As is usual for dynamic programming on tree decompositions, we group partial solution by *characteristics*. Given a partial solution S for node v_T of the nice tree decomposition (with associated bag X_{v_T} and subgraph $G[v_T]$), its characteristic (ℓ, S', f) consists of the size of the solution $\ell = |S|$, its intersection with the bag $S' = X_{v_T} \cap S$, together with a function $f : X_{v_T} \rightarrow \{0, 1, \dots, s\}$ such that $f(v) = |\{u \in S \mid u \in N(v)\}|$, which, for each vertex v in the bag X_{v_T} , tells us how many of its neighbors are in the partial solution.

For each characteristic, we store the minimum cost of a partial solution with that characteristic, which we denote by $c(\ell, S', f)$. Next, we show how to recursively compute for each type of node in a nice tree decomposition the set of characteristics of a partial solutions, and for each such characteristic, the minimum number of 3-paths not covered by a partial solution with that characteristic.

Leaf: If $v_T \in V(T)$ is a leaf node, then $X_{v_T} = \{v\}$ for some $v \in V(G)$. Then there are exactly two partial solutions for $G[v_T]$: the empty partial solution, which has characteristic $(0, \emptyset, f)$ where $f(v) = 0$ and the partial solution that includes v , which has characteristic $(1, \{v\}, f)$, where $f(v) = 0$. In both cases, $c(0, \emptyset, f) = c(1, \{v\}, f) = 0$.

Introduce: Suppose that $v_T \in V(T)$ is an introduce node, and v is the vertex being introduced. Let (ℓ, S', f) be a characteristic for the child node of v_T . In the partial solutions (for the child node) with this characteristic, we may (if $\ell < s$) choose to either add the vertex v or not. In the case where we add v , the corresponding partial solutions have characteristic $(\ell + 1, S' \cup \{v\}, f')$, where $f'(v) = |\{u \in S' \mid u \in N(v)\}|$, and, if $u \neq v$ and $u \notin N(v)$, $f'(u) = f(u)$, and, if $u \neq v$ and $u \in N(v)$, $f'(u) = f(u) + 1$. In the case where we do not add v , the corresponding partial solutions will have characteristic (ℓ, S', f') , where $f'(v) = |\{u \in S' \mid u \in N(v)\}|$, and, if $u \neq v$, $f'(u) = f(u)$. Since v is not adjacent to any vertex in $G[v_T] \setminus X_{v_T}$, the cost of these partial solutions remains unchanged. Note, however, that taking two partial solutions with distinct characteristics may end up having the same characteristic after vertex v is introduced. In this case, we should take the cost (for the new characteristic) to be the minimum of the costs for the original partial solutions.

Forget: Suppose that $v_T \in V(T)$ is a forget node, and v is the vertex being forgotten. Let (ℓ, S', f) be a characteristic for the child node of v_T . If S is a

partial solution with this characteristic, then, viewed as a partial solution with respect to node v_T , it will have characteristic $(\ell, S' \setminus \{v\}, f')$, where f' is the restriction of f to the domain $S' \setminus \{v\}$. If $v \notin S'$ and $f(v) < \text{deg}(v)$, then the cost of this partial solution increases by $\frac{1}{2}(\text{deg}(v) - f(v))(\text{deg}(v) - f(v) - 1)$, otherwise it remains unchanged. As before, since multiple characteristics for the child node may end up having the same characteristic in v_T , and we should take the new cost of the characteristic to be the minimum of the updated costs.

Join: Suppose that $v_T \in V(T)$ is a join node, and v_T^1 and v_T^2 are its children. Let (ℓ_1, S'_1, f_1) (resp., (ℓ_2, S'_2, f_2)) be a characteristic for v_T^1 (resp., v_T^2). Assume that $S'_1 = S'_2$, which we henceforth denote simply by S' , and that $\ell_1 + \ell_2 - |S| \leq s$. If we take the union of partial solutions, S_1 relative to v_T^1 with characteristic (ℓ_1, S', f_1) and S_2 relative to v_T^2 with characteristic (ℓ_2, S', f_2) , then we obtain a new partial solution (relative to v_T) with characteristic $(\ell_1 + \ell_2 - |S'|, S', f')$, where $f'(v) = f_1(v) + f_2(v) - |\{u \in S' \mid u \in N(v)\}|$. Since $V(G[v_T^1]) \cap V(G[v_T^2]) = X_{v_T}$, in $G[v_T]$, the degree of a vertex $v \in V(G[v_T]) \setminus (X_{v_T} \cup S)$ is equal to its degree in (the subgraph of G induced by) $G[v_T^1] \setminus S$ (resp., $G[v_T^2] \setminus S$) if $v \in V(G[v_T^1]) \setminus S$ (resp., $v \in V(G[v_T^2]) \setminus S$). Therefore, the cost of this new partial solution is equal to the sum of the costs of the partial solutions S_1 and S_2 . Since once again, multiple (combinations of) characteristics for the child nodes may give rise to the same characteristic for v_T , we can find the minimum cost of a partial solution for a given characteristic by taking the minimum over all (combinations of) characteristics for the child nodes.

For any node, there are at most $(s+1)^{tw+2}2^{tw+1}$ characteristics. The running time is dominated by the time taken for a join node, which is $O((s+1)^{2tw+4} \cdot 4^{tw+1})$. Since we can assume that our tree decomposition has at most $O(n)$ nodes, we obtain a $O((s+1)^{2tw+4} \cdot 4^{tw} \cdot n)$ -time algorithm. This assumes a tree decomposition is given as part of the input. A tree decomposition can be computed in $2^{O(tw^3)}n$ time [3], or a 5-approximate tree decomposition can be computed in time $O(1)^{tw}n$ [4].

Theorem 6. *MaxP₃VC on n -vertex graphs of treewidth tw can be solved in $O((s+1)^{2tw+4} \cdot 4^{tw} \cdot n)$ time, where s is the prescribe size of the 3-path vertex cover.*

References

1. Apollonio, N., Simeone, B.: The maximum vertex coverage problem on bipartite graphs. *Dis. Appl. Math.* **165**, 37–48 (2014)
2. Betzler, N., Niedermeier, R., Uhlmann, J.: Tree decompositions of graphs: saving memory in dynamic programming. *Dis. Optim.* **3**, 220–229 (2006)
3. Hans, L.: A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.* **25**(6), 1305–1317 (1996)
4. Bodlaender, H.L., Drange, P.G., Dregi, M.S., Fomin, F.V., Lokshtanov, D., Pilipczuk, M.: A $O(c^k n)$ 5-approximation algorithm for treewidth. *SIAM J. Comput.* **45**(2), 317–378 (2016)

5. Brešar, B., Kardoš, F., Katrenič, J., Semanišin, G.: Minimum k -path vertex cover. *Dis. Appl. Math.* **159**(12), 1189–1195 (2011)
6. Camby, E.: Connecting hitting sets and hitting paths in graphs. Ph.D. thesis, Doctoral Thesis (2015)
7. Caskurlu, B., Mkrtchyan, V., Parekh, O., Subramani, K.: On partial vertex cover and budgeted maximum coverage problems in bipartite graphs. In: *IFIP International Conference on Theoretical Computer Science*, pp. 13–26. Springer, Heidelberg (2014)
8. Courcelle, B.: Graph rewriting: an algebraic and logic approach. In: *Handbook of Theoretical Computer Science*, vol. B, pp. 193–242 (1990)
9. Gonzalez, T.F.: Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.* **38**, 293–306 (1985)
10. Karp, R.: Reducibility among combinatorial problems. In: *Complexity of Computer Computations*, pp. 85–103 (1972)
11. Katrenič, J.: A faster FPT algorithm for 3-path vertex cover. *Inf. Process. Lett.* **116**(4), 273–278 (2016)
12. Li, X., Zhang, Z., Huang, X.: Approximation algorithms for minimum (weight) connected k -path vertex cover. *Dis. Appl. Math.* **205**, 101–108 (2016)
13. Papadimitriou, C.H., Yannakakis, M.: The complexity of restricted spanning tree problems. *J. ACM* **29**(2), 285–309 (1982)
14. Jianhua, T., Jin, Z.: An FPT algorithm for the vertex cover P4 problem. *Dis. Appl. Math.* **200**, 186–190 (2016)
15. Jianhua, T., Zhou, W.: A factor 2 approximation algorithm for the vertex cover P3 problem. *Inf. Process. Lett.* **111**(14), 683–686 (2011)
16. Jianhua, T., Zhou, W.: A primal-dual approximation algorithm for the vertex cover P3 problem. *Theor. Comput. Sci.* **412**(50), 7044–7048 (2011)
17. Xiao, M., Kou, S.: Exact algorithms for the maximum dissociation set and minimum 3-path vertex cover problems. *Theor. Comput. Sci.* **657**, 86–97 (2017)
18. Yannakakis, M.: Node-deletion problems on bipartite graphs. *SIAM J. Comput.* **10**, 310–327 (1981)
19. Zhang, Z., Li, X., Shi, Y., Nie, H., Zhu, Y.: PTAS for minimum k -path vertex cover in ball graph. *Inf. Process. Lett.* **119**, 9–13 (2017)

On the Parallel Parameterized Complexity of the Graph Isomorphism Problem

Bireswar Das, Murali Krishna Enduri^(✉), and I. Vinod Reddy

IIT Gandhinagar, Gandhinagar, India
{bireswar, endurimuralikrishna, reddy_vinod}@iitgn.ac.in

Abstract. In this paper, we study the parallel and the space complexity of the graph isomorphism problem (GI) for several parameterizations.

Let $\mathcal{H} = \{H_1, H_2, \dots, H_i\}$ be a finite set of graphs where $|V(H_i)| \leq d$ for all i and for some constant d . Let \mathcal{G} be an \mathcal{H} -free graph class i.e., none of the graphs $G \in \mathcal{G}$ contain any $H \in \mathcal{H}$ as an induced subgraph. We show that GI parameterized by vertex deletion distance to \mathcal{G} is in a parameterized version of AC^1 , denoted Para-AC^1 , provided the colored graph isomorphism problem for graphs in \mathcal{G} is in AC^1 . From this, we deduce that GI parameterized by the vertex deletion distance to cographs is in Para-AC^1 .

The parallel parameterized complexity of GI parameterized by the size of a feedback vertex set remains an open problem. Towards this direction we show that the graph isomorphism problem is in Para-TC^0 when parameterized by vertex cover or by twin-cover.

Let \mathcal{G}' be a graph class such that recognizing graphs from \mathcal{G}' and the colored version of GI for \mathcal{G}' is in logspace (L). We show that GI for bounded vertex deletion distance to \mathcal{G}' is in L. From this, we obtain logspace algorithms for GI for graphs with bounded vertex deletion distance to interval graphs and graphs with bounded vertex deletion distance to cographs.

1 Introduction

Two graphs $G = (V_g, E_g)$ and $H = (V_h, E_h)$ are said to be *isomorphic* if there is a bijection $f : V_g \rightarrow V_h$ such that for all pairs $\{u, v\} \in \binom{V_g}{2}$, $\{u, v\} \in E_g$ if and only if $\{f(u), f(v)\} \in E_h$. Given a pair of graphs as input the problem of deciding if the two graphs are isomorphic is known as the *graph isomorphism problem* (GI). Whether this problem has a polynomial-time algorithm is one of the outstanding open problem in the field of algorithms and complexity theory. It is in NP but very unlikely to be NP-complete as it is in $\text{NP} \cap \text{coAM}$ [7]. Recently Babai [4] designed a quasi-polynomial time algorithm for GI improving the best previously known runtime $2^{O(\sqrt{n \log n})}$ [2, 37]. However, efficient algorithms for GI have been discovered for various restricted classes of graphs e.g., planar graphs [26],

M. K. Enduri — Supported by Tata Consultancy Services (TCS) research fellowship.

bounded degree graphs [31], bounded genus graphs [33], bounded tree-width graphs [6] etc.

For restricted classes of graphs the complexity of GI has been studied more carefully and finer complexity classifications within P have been done. Lindell [29] gave a deterministic logspace algorithm for isomorphism of trees. In the recent past, there have been many logspace algorithms for GI for restricted classes of graphs e.g., $K_{3,3}$ or K_5 minor free graphs [19], planar graphs [18], bounded tree-depth graphs [16], bounded tree-width graphs [21] etc. On the other hand parallel isomorphism algorithms have been designed for graphs with bounded eigenvalue multiplicity [3], bounded color class graphs [32] etc.

The graph isomorphism problem has been studied in the parameterized framework for several graph classes with parameters such as the tree-depth [8], the tree-distance width [36], the connected path distance width [34] and recently the tree-width which corresponds to a much larger class [30]. A more detailed list of FPT algorithms for GI in parameterized setting can be found in [9].

While there are many results on the parallel or the logspace complexity of problems in the parameterized framework [20], very little is known in this direction for GI. The parameterized analogues of classical complexity classes have also been studied in [12, 22, 23]. The class $\text{Para-}\mathcal{C}$ is the family of parameterized problems that are in \mathcal{C} after a pre-computation on the parameter, where \mathcal{C} is a complexity class. In this paper we study the graph isomorphism problem from a parameterized space and parallel complexity perspective. Recently Chandoo [14] showed that GI for circular-arc graphs is in Para-L when parameterized by the cardinality of an obstacle set.

Since the graph isomorphism problem parameterized by tree-width has a logspace [21] as well as a separate FPT algorithm [30] it is natural to ask if we can design a *parameterized parallel* algorithm for this problem. In fact, the parallel complexity of GI parameterized by the well known but weaker parameter feedback vertex set number (FVS) is also unknown. We make some progress in this direction by showing that GI parameterized by the size of a vertex cover, which is a weaker parameter than the FVS, is parallelizable in the parameterized setting.

Let \mathcal{G} be a graph class characterized by a finite set of forbidden induced subgraphs (see Sect. 3 for the formal definition). Kratsch et al. [28] gave an FPT algorithm for GI parameterized by the distance to \mathcal{G} by taking a polynomial time colored graph isomorphism algorithm for graphs in \mathcal{G} as a subroutine. In Sect. 3, we show that the result of [28] is parallelizable in the parameterized framework. More precisely, we give a Para-AC^1 algorithm for this problem. As a consequence, observe that GI parameterized by the distance to cographs is in Para-AC^1 .

Using bounded search tree method we also design a parallel recognition algorithm for graphs parameterized by the distance to \mathcal{G} . One would ask if the problem is in Para-L using the same method as in [12] and [20]. However, the recent corrigendum Cai et al. [13] suggests that this may need completely new ideas.

In the above mentioned parallel analogue of the result by Kratsch et al. [28], \mathcal{G} is a class of graphs characterized by a finite set of forbidden induced subgraphs. Instead of that if we take \mathcal{G} to be the set of bounded tree-width graphs then the parallel parameterized complexity is again open. Note that the analogous preconditions of the theorem by Kratsch et al. [28] in this scenario is met by the logspace GI algorithm for bounded tree-width graphs by Elberfeld et al. [21]. In fact, the problem is open even when \mathcal{G} is just the set of forests because this is the same problem: GI parameterized by feedback vertex set number. We study the graph isomorphism problem for *bounded* distance to any graph class \mathcal{G} under reasonable assumptions: the colored version of GI for the class \mathcal{G} and the recognition problem for \mathcal{G} are in L. We give a logspace isomorphism algorithm for such classes of graphs.

In Sect. 4, we show that GI is in Para-TC^0 when parameterized by the vertex cover number. By using the recognition algorithm for graphs parameterized by the vertex cover number due to [5], we first design a recognition algorithm for graphs parameterized by twin-cover number. We then prove that the graph isomorphism problem parameterized by twin-cover is in Para-TC^0 .

2 Preliminaries

The basic definitions and notations of standard complexity classes are from [1] and the definitions of parameterized versions of complexity classes are from [12, 22, 35]. A *parameterized problem* is pair (\mathcal{Q}, k) of a language $\mathcal{Q} \subseteq \Sigma^*$ and a parameterization $k : \Sigma^* \rightarrow \mathbb{N}$ that maps input instances to natural numbers, their parameter values¹. The class $\text{Para-}\mathcal{C}$ is defined to be the family of problems that are in \mathcal{C} after a precomputation on the parameter where \mathcal{C} is a complexity class.

Definition 1 [22]. *For a complexity class \mathcal{C} , a parameterized problem (\mathcal{Q}, k) belongs to the para class $\text{Para-}\mathcal{C}$ if there is an alphabet Π , a computable function $\pi : \mathbb{N} \rightarrow \Pi^*$ and a language $A \subseteq \Sigma^* \times \Pi^*$ with $A \in \mathcal{C}$ such that for all $x \in \Sigma^*$ we have $x \in \mathcal{Q} \Leftrightarrow (x, \pi(k(x))) \in A$.*

If the complexity class \mathcal{C} is L then we get the complexity class Para-L . The following equivalent definition of Para-L is convenient when designing Para-L algorithms.

Definition 2 [22]. *A parameterized problem (\mathcal{Q}, k) over Σ is in Para-L if there is function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that the question $x \in \mathcal{Q}$ can be decided within space $f(k) + O(\log |x|)$.*

The parameterized parallel complexity classes are defined by using the basic complexity classes in place of L in above and basic gates (*AND* and *OR* gates) as follows [35]:

¹ Often we write k in stead of $k(x)$.

Para-ACⁱ (Para-TCⁱ): The class of languages that are decidable via family of circuits over basic gates (resp. together with threshold gates) with unbounded fan-in, size $O(f(k)n^{O(1)})$, and depth $O(f(k) + \log^i n)$ if $i > 0$ and depth $O(1)$ if $i = 0$. From the definition of Para- \mathcal{C} , we know that for two complexity classes \mathcal{C} and \mathcal{C}' , $\mathcal{C} \subseteq \mathcal{C}'$ if and only if $\text{Para-}\mathcal{C} \subseteq \text{Para-}\mathcal{C}'$ [5]. Hence we have the following relation between complexity classes $\text{Para-AC}^0 \subsetneq \text{Para-TC}^0 \subseteq \text{Para-L} \subseteq \text{Para-AC}^1$. There exists a circuit class $\text{Para-AC}^{0\uparrow}$ in between Para-AC^0 and Para-AC^1 which is strictly more powerful than Para-AC^0 . The definition of $\text{Para-AC}^{0\uparrow}$ is as follows.

Definition 3 [5]. *Para-AC^{0\uparrow} is a class of languages that are decidable via family of circuits over basic gates with unbounded fan-in, size $O(f(k)n^{O(1)})$, and depth $g(k)$ where f and g are computable functions.*

The depth of the circuits in this class is bounded by a function that depends only on the parameter. We have $\text{Para-AC}^0 \subsetneq \text{Para-AC}^{0\uparrow} \subseteq \text{Para-AC}^1$ and $\text{Para-AC}^0 \subsetneq \text{Para-L} \subseteq \text{Para-AC}^1$. We do not know relation between $\text{Para-AC}^{0\uparrow}$ and Para-L . The computational versions of all the above circuit classes can be defined in the usual manner by having multiple output gates.

In this paper, the graphs we consider are undirected and simple. For a graph $G = (V, E)$, let $V(G)$ and $E(G)$ denote the vertex set and edge set of G respectively. An edge $\{u, v\} \in E(G)$ is denoted as uv for simplicity. For a subset $S \subseteq V(G)$, the graph $G[S]$ denotes the subgraph of G induced by the vertices of S . We use notation $G \setminus S$ to refer the graph obtained from G after removing the vertex set S . For a vertex $u \in V(G)$, $N(u)$ denotes the set of vertices adjacent to u and $N[u] = N(u) \cup \{u\}$. For a set $X \subseteq V(G)$, $N(X)$ denoted as $\cup_{v \in X} N(v)$.

In this paper we study problems similar to the graph modification problems where given a graph G , and a graph class \mathcal{G} the task is to apply some graph operations (such as vertex or edge deletions) on G to get a graph in \mathcal{G} . For example, if \mathcal{G} is the class of edgeless graphs then the number of vertices to be deleted from graph G to make it edgeless is the vertex cover problem. For a graph class \mathcal{G} , the *distance to \mathcal{G}* of a graph G is the minimum number of vertices to be deleted from G to get a graph in \mathcal{G} . For a positive integer k , we use $\mathcal{G} + kv$ to denote the family of graphs such that each graph in this family can be made into a graph in \mathcal{G} by removing at most k vertices.

Cographs are P_4 -free graphs i.e., they do not contain any induced paths on four vertices. *Interval graphs* are the intersection graphs of a family of intervals on the real line. A graph is a *threshold graph* if it can be constructed recursively by adding an isolated vertex or a universal vertex.

The parameterized VERTEX COVER problem has input a graph G and a positive integer k . The problem is to decide the existence of a vertex set $X \subseteq V(G)$ of size at most k such that for every edge $uv \in E(G)$, either $u \in X$ or $v \in X$. A *minimal vertex cover* of a graph is a vertex cover that does not contain another vertex cover.

Definition 4. Let G be a graph. The set $X \subseteq V(G)$ is said to be twin-cover of G if for every edge $uv \in E(G)$ either (a) $u \in X$ or $v \in X$, or (b) u and v are twins².

An edge between a pair of twins is called a *twin edge*. The graph G' obtained by removing a twin-cover from G is a disjoint collection of cliques [24].

A *kernel* for a parameterized problem \mathcal{Q} is an algorithm which transforms an instance (I, k) of \mathcal{Q} to an equivalent instance (I', k') in polynomial time such that $k' \leq k$ and $|I'| \leq f(k)$ for some computable function f . For more details on parameterized complexity see [20].

In this paper, a coloring of a graph is just a mapping of the vertices of a graph to a set of colors, and it need not be proper.

Definition 5. The colored graph isomorphism problem is to decide the existence of a color preserving isomorphism between a pair of colored graphs $G = (V, E)$ and $G' = (V', E')$, i.e., there exists a bijection mapping $\varphi : V \rightarrow V'$, satisfying the following conditions: (1) $(u, v) \in E \Leftrightarrow (\varphi(u), \varphi(v)) \in E'$ for all $u, v \in V$ (2) $color(v) = color(\varphi(v))$ for all $v \in V$.

Due to the space limitation, some of the proofs can be found in the full version of the paper [17].

3 GI for Distance to a Graph Class is in Para-AC¹

In this Section, first we give a generic method to solve GI for graphs from $\mathcal{G} + kv$ in Para-L provided there is a logspace colored GI algorithm for graphs in \mathcal{G} and a Para-L algorithm for enumerating vertex deletion sets.

Theorem 1. Let \mathcal{G} be a any graph class. Suppose enumerating all the vertex deletion sets of $\mathcal{G} + kv$ is in Para-L and the colored graph isomorphism problem for graphs from \mathcal{G} is in L. Then the graph isomorphism problem for graphs from $\mathcal{G} + kv$ is in Para-L.

Proof. Let $\mathcal{A}_{\mathcal{I}}$ be a logspace algorithm to check whether two given input colored graphs G_1 and G_2 from \mathcal{G} are isomorphic. We assume that graphs G_1 and G_2 are at a distance at most k from \mathcal{G} . If G_1 and G_2 belong to \mathcal{G} then use the algorithm $\mathcal{A}_{\mathcal{I}}$ to check the isomorphism between G_1 and G_2 . Otherwise we consider a vertex deletion set $S \subseteq V(G_1)$ of minimum size (say s) such that $G_1 \setminus S \in \mathcal{G}$ and all possible vertex deletion sets S_1, S_2, \dots, S_m of size s for G_2 such that $G_2 \setminus S_i \in \mathcal{G}$ for all $i \in [m]$ given as input. Notice that $m \leq f(k)$.

For each $i \in [m]$, the algorithm iterates over all possible isomorphisms between $G[S]$ to $G[S_i]$, and tries to extend them to isomorphisms from G_1 to G_2 with the help of the colored graph isomorphism algorithm $\mathcal{A}_{\mathcal{I}}$ applied on some colored versions of $G_1 \setminus S$ and $G_2 \setminus S_i$, where the colors of the vertices are determined by their neighbors in the corresponding deletion set. A crucial

² Two vertices u and v are *twins* if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$.

observation is that any bijective mapping from S to S_i can be viewed as a string in $[s]^s$ and can be encoded as a string of length $O(k \log k)$. The string is $x_1 \cdots x_s$ encodes the map that sends the i th vertex in S to the x_i th vertex in S_i .

For all i algorithm iterates over all $s!$ bijective mappings from S to S_i using string of length $O(s \log s)$. Next it checks whether the bijective mapping is actually an isomorphism from $G_1[S]$ to $G_2[S_i]$. For each isomorphism φ from S to S_i , we need to check whether this isomorphism can be extended to an isomorphism from $G_1 \setminus S$ to $G_2 \setminus S_i$ by using algorithm $\mathcal{A}_{\mathcal{T}}$. We color the vertices of $G_1 \setminus S$ according to their neighbourhood in S . Two vertices of $G_1 \setminus S$ get same color if they have the same neighbourhood in S . A vertex u in $G_1 \setminus S$ and a vertex v in $G_2 \setminus S_i$ will get same color if $\varphi(N(u) \cap S) = N(v) \cap S_i$. We query algorithm $\mathcal{A}_{\mathcal{T}}$ with input the graphs $G_1 \setminus S$ and $G_2 \setminus S_i$ colored as above. If the algorithm $\mathcal{A}_{\mathcal{T}}$ says ‘yes’ then $G_1 \cong G_2$ and the algorithm accepts the input. Otherwise it tries the next isomorphism from S to S_i . If for all i and all isomorphisms from S to S_i , the algorithm $\mathcal{A}_{\mathcal{T}}$ rejects then we conclude that $G_1 \not\cong G_2$ and the algorithm rejects the input.

We note few more details of the algorithm to demonstrate that it uses small space. The enumeration over the S_i ’s can be done using a $\log m$ bit counter. To check if two vertices u in $G_1 \setminus S$ and v in $G_2 \setminus S_2$ have same color in logspace we can inspect each vertex in G_1 , find out if it is in S , find out if it is a neighbour of u , and check if its image under φ is a neighbour of v . This needs constantly many counters. □

Next we give a $\text{Para-AC}^{0\uparrow}$ recognition algorithm for graphs parameterized by the distance to a graph class \mathcal{G} by using the bounded search tree technique, where \mathcal{G} is characterized by finitely many forbidden induced subgraphs.

Definition 6 [28]. *A class \mathcal{G} of graphs is characterized by finitely many forbidden induced subgraphs if there is a finite set of graphs $\mathcal{H} = \{H_1, H_2, \dots, H_l\}$ such that a graph G is in \mathcal{G} if and only if G does not contain H_i as an induced subgraph for any $i \in \{1, 2, \dots, l\}$.*

Let \mathcal{G} and \mathcal{H} be classes as defined above. We use the bounded search tree technique [11, 20] to find a set S of size at most k such that $G \setminus S \in \mathcal{G}$. In this method we can compute all deletion sets of size at most k . Let d be the size of the largest forbidden induced subgraph in \mathcal{H} . The algorithm constructs a tree T as follows. The root of the tree is labelled with the empty set. It finds a forbidden induced subgraph $H_i \in \mathcal{H}$ of size at most d in G . Any vertex deletion set S must contain a vertex of H_i . We add $|V(H_i)|$ many children to the root labelled with vertices of H_i . In general if a node is labelled with a set P , then we find a forbidden induced subgraph H_j in $G \setminus P$ and create $|V(H_j)|$ many children for the node labeled P and label each child with $P \cup \{v_i\}$, where $v_i \in H_j$. If there exists a node labeled with a set S in T of size at most k such that $G \setminus S \in \mathcal{G}$, then S is a required vertex deletion set. From this, we also know that there are at most d^k minimal vertex deletion sets of size at most k . Using the same process we can also find all the minimal vertex deletion sets of size at most k .

Cai et al. [12] implemented bounded search tree method and kernelization to find the vertex cover in Para-L in 1997. However, the implementation of bounded search tree method in Para-L was reported to have some errors [13]. Thus, this paper seems to give the first implementation of bounded search tree method in Para-AC^{0†}. Let us recall from Sect. 2, that there is no known relation between Para-AC^{0†} and Para-L.

Lemma 1. *Let \mathcal{G} be a class of graphs characterized by finitely many forbidden induced subgraphs $\mathcal{H} = \{H_1, H_2, \dots, H_l\}$ with $|V(H_i)| \leq d$ for all $1 \leq i \leq l$ where d is a constant. On input a graph G , the problem of computing all vertex deletion sets of size at most k is in Para-AC^{0†} where k is the parameter.*

Proof. The idea to implement the bounded search tree method in Para-AC^{0†} is as follows:

Consider the set of all subsets of size at most d that induce a forbidden subgraph in G . We order these subsets lexicographically to obtain a list $\mathcal{L} = A_1, \dots, A_m$ where for each i , $G[A_i]$ is isomorphic to some graph in \mathcal{H} . Notice that $m = O(n^d)$. The list \mathcal{L} can be computed in Para-AC^{0†} by first producing all subsets of $V(G)$ of size at most d and then keeping only those that induce a subgraphs isomorphic to some H in \mathcal{H} . Observe that any vertex deletion set must contain at least one vertex from each A_i for all i . The algorithm uses all strings $\Gamma = \gamma_1 \dots \gamma_k \in [d]^k$ in parallel to pick the vertex deletion sets S of size at most k as follows: Let us concentrate on the part of the circuit that processes a particular string $\Gamma = \gamma_1 \dots \gamma_k$. Initially the deletion set S is empty. The algorithm puts the γ_i th vertex (in lexicographic order) of A_1 in S if $|A_1| \geq \gamma_1$. If $|A_1| < \gamma_1$ the computation ends in this part of the circuit. Suppose the algorithm has already picked i vertices using $\gamma_1 \dots \gamma_i$. It picks the $(i + 1)$ th vertex using γ_{i+1} . To do so it picks the first set A_j in the list \mathcal{L} such that $A_j \cap S = \phi$ (if $A_j \cap S \neq \phi$ we say that A_j is ‘hit’ by S). Then it puts the γ_{i+1} th vertex of A_j in S if $|A_j| \geq \gamma_{i+1}$. Otherwise the computation ends in the part processing Γ . If on or before reaching γ_k we have obtained a set S such that $A_j \cap S \neq \phi$ for all j , the algorithm has successfully found a vertex deletion set. We say that the algorithm is in *phase i* if it processing γ_i .

To see that the algorithm can be implemented in Para-AC^{0†}, we just need to observe that in each phase the algorithm has to maintain the list of sets in \mathcal{L} that are not yet hit by S . The depth of the circuit is $O(k)$ and the total size is $d^k \text{poly}(n)$. □

We implemented the bounded search tree method in Para-AC^{0†}. This implementation can be used not only to recognize the graph class defined in the Definition 6 but also, as we can show, for designing Para-AC^{0†} algorithms for the problems RESTRICTED ALTERNATING HITTING SET and WEIGHT $\leq k$ q -CNF SATISFIABILITY (for more details see in [17]).

The next theorem is obtained by replacing the complexity class Para-L by Para-AC¹ in Theorem 1. The proof of the theorem uses similar ideas and the implementation is easier. Moreover, because of Lemma 1 we do not have to assume the existence of an algorithm that outputs all the vertex deletion sets.

Theorem 2. *Let \mathcal{G} be a class of graphs characterized by finitely many forbidden induced subgraphs $\mathcal{H} = \{H_1, H_2, \dots, H_l\}$ with $|V(H_i)| \leq d$ for all $1 \leq i \leq l$ where d is a constant. Suppose the colored graph isomorphism problem for graphs from \mathcal{G} is in AC^1 . Then the graph isomorphism problem for graphs from $\mathcal{G} + kv$ is in Para-AC^1 .*

Corollary 1. *The graph isomorphism problem parameterized by the distance to cographs is in Para-AC^1 .*

Proof. Recall that cographs are graphs without any induced P_4 . The colored graph isomorphism for cographs was shown to be in L using logspace algorithm to find the modular decomposition [25]. From this along with Theorem 2 and Lemma 1, we deduce that the graph isomorphism problem for distance to cographs is in Para-AC^1 . \square

As a consequence of the above corollary, we can also solve graph isomorphism problem for some of the other graph classes e.g., distance to cluster (disjoint union of cliques), distance to threshold graphs in Para-AC^1 by using the generalized meta Theorem 2.

For larger parameters like vertex-cover, distance to clique and twin-cover, we can get better complexity theoretic results which we discuss in the following section.

4 GI Parameterized by Vertex Cover is in Para-TC^0

In this section we give a parameterized parallel algorithm for GI parameterized by vertex cover. Sam Buss [10] showed that VERTEX COVER admits a polynomial kernel. Based on this kernelization result, Cai et al. [12], Elberfeld et al. [22] and Bannach et al. [5] showed that VERTEX COVER is in Para-L , Para-TC^0 and Para-AC^0 respectively. These methods not only determines the existence of a vertex cover of size at most k but can also output all vertex covers of size at most k in Para-AC^0 . We give a brief overview of the procedure to enumerate all vertex covers of size at most k by using kernelization method given in [5, 12, 22].

Observe that any vertex of degree more than k must belong to any vertex cover of a given graph G . For the graph $G = (V, E)$, consider the set $V_H = \{v \in V(G) \mid d(v) > k\}$. If $|V_H|$ is more than k then we declare that there is no k sized vertex cover. Let us assume $|V_H| = b$. Consider the set $V_L = \{v \in V(G) \mid d(v) \leq k \text{ and } N(v) \setminus V_H \neq \emptyset\}$ of vertices that have at least one neighbour outside V_H . Notice that none of the edges in $G[V_L]$ are covered by V_H . Let S' be a vertex cover of $G[V_L]$. It is easy to see that $V_H \cup S'$ forms a vertex cover of G . On the other hand if S is a vertex cover of G then $V_L \cap S$ is a vertex cover of $G[V_L]$. If the cardinality of V_L is more than $(k - b)(k + 1)$ then reject (because the graph induced by vertices V_L with $k - b$ vertex cover and all vertices degree bounded by k has no more than $(k - b)(k + 1)$ vertices). So the cardinality of V_L is not more than $(k - b)(k + 1)$. We can use the best known vertex cover algorithm [15] to find the $(k - b)$ vertex cover on the sub graph induced by vertices V_L . Elberfeld et al. [22] pointed that the parallel steps of this process are the following:

- i. Checking whether the vertex belongs to V_H .
- ii. Checking whether $|V_H|$ at most k .
- iii. Checking whether $|V_L|$ at most $k(k + 1)$.
- iv. Computing the induced subgraph $G[V_L]$ from G .

The computation of above steps can be implemented by Para-AC^0 circuits [5]. The above process finds all vertex covers of size at most k by enumerating all the $2^{|V_L|}$ possible binary strings on length $|V_L|$.

Theorem 3. *The graph isomorphism problem parameterized by vertex cover is in Para-TC^0 .*

Proof. Given two input graphs G and H with vertex cover of size at most k , we need to test if G to H are isomorphism in Para-TC^0 . Using the kernelization method of Bannach et al. [5] we can recognize whether these two graphs have same sized vertex covers or not. For the graph G we find a minimal vertex cover S of size at most k and for graph H we find all minimal vertex covers S_1, S_2, \dots, S_m , each of size at most k . Notice that m is at most 2^k . We know that if $G \cong H$ then $G[S] \cong H[S_i]$ for some $1 \leq i \leq m$. We try all isomorphisms from the minimal vertex cover S of G to each minimal vertex cover S_i of H . Suppose $G[S] \cong H[S_i]$ via φ . We need to extend this isomorphism from the independent set $G \setminus S$ to $H \setminus S_i$. There are at most $k!$ isomorphisms between $G[S]$ to $H[S_i]$. The algorithm processes all pairs (S, S_i) and all the isomorphisms in parallel.

For each isomorphism φ , we need to check whether this φ can be extended to an isomorphism between $G \setminus S$ to $H \setminus S_i$. We partition the vertices of the graph $G \setminus S$ into at most 2^k sets (called ‘types’) based on their neighborhood in S . For each $U \subseteq S$ let $T_G(U, S) = \{u \in G \setminus S \mid N(u) = U\}$. It is not hard to see that $G \cong H$ if and only if there is a minimal vertex cover S_i of H and an isomorphism φ from $G[S]$ to $H[S_i]$ such that for each $U \subseteq S$, $|T_G(U, S)| = |T_H(\varphi(U), S_i)|$. The problem of testing whether G is isomorphic to H reduces to counting the number of vertices in each type. We represent each type using an n -length binary string, where i^{th} entry is one if v_i belongs to that type and zero otherwise. Since the BIT COUNT^3 problem is in TC^0 , counting the number of vertices in a type can be implemented using a TC^0 circuit. In summary, for each S_i and each isomorphism between $G[S]$ and $H[S_i]$, and for each $U \subseteq S$ we check whether $|T_G(U, S)| = |T_H(\varphi(U), S_i)|$. This completes the proof. \square

Corollary 2. *The graph isomorphism problem is in Para-TC^0 when parameterized by the distance to clique.*

Proof. We apply Theorem 3 to the complements of the input graphs. \square

Corollary 3. *The graph isomorphism problem parameterized by the size of a twin-cover is in Para-TC^0 .*

³ Counting the number one’s in n length binary string.

Proof. To find the twin-cover, we first remove all the twin edges and then compute a vertex cover of size at most k in the resulting graph as was done in [24]. The first step runs through all edges and deletes an edge if it is a twin edge. Next it finds a vertex cover in the resulting graph which can be done in Para-AC^0 [5]. Thus, computing all the twin-covers can be done in Para-AC^0 .

Now we describe the process of testing isomorphism. The idea for testing isomorphism of the input graphs parameterized by the size of a twin-cover is similar to that in the proof of Theorem 3. Let S_1 be a fixed twin-cover in G_1 and S_2 be a twin-cover in G_2 of same size. The algorithm processes all such (S_1, S_2) pairs in parallel. First fix an isomorphism (say σ) from S_1 to S_2 and try to extend it to $G_1 \setminus S_1$ to $G_2 \setminus S_2$. Again, all such isomorphisms are processed in parallel. We know that the graph $G \setminus S$ obtained by removing a twin-cover S from G is a disjoint collection of cliques. Any two vertices in a clique C have same neighbourhood in G i.e., if $u, v \in C$ then $N[u] = N[v]$. Thus, the ‘type’ of a clique is completely determined by the neighbourhood of any of the vertices in the vertex deletion set, and the size of the clique. Formally, with respect to the isomorphism σ , a clique C_{g_1} in $G_1 \setminus S_1$ and a clique C_{g_2} in $G_2 \setminus S_2$ have same *type* if (1) $|V(C_{g_1})| = |V(C_{g_2})|$ and (2) $\sigma(N(C_{g_1})) = N(C_{g_2})$. The algorithm needs to check that the number of cliques in each type is same in both the graphs. This problem can again be reduced to instances of the BIT COUNT problem.

It is easy to see that, the above process can be implemented in Para-TC^0 . \square

5 Logspace GI Algorithms for Bounded Distance to Graph Classes

In this Section, we show that for fixed k GI for graphs in $\mathcal{G} + kv$ is in L if the colored GI for graphs in \mathcal{G} is in L where \mathcal{G} is a graph class. From this result we obtain that GI for *cographs* + kv and *interval* + kv graphs is in L . Note that these results are not in the parameterized complexity theory framework. The proof of the following theorem can be found in [17].

Theorem 4. *Let k be a fixed and \mathcal{G} be a class of graphs. Suppose the problem of deciding if a given graph is in \mathcal{G} and the colored graph isomorphism problem for graphs in \mathcal{G} is in L . Then the graph isomorphism problem for graphs from $\mathcal{G} + kv$ is in L .*

Suppose graph class \mathcal{G} is define as in Definition 6. It is not hard to see the problem of deciding if a graph G is in a class \mathcal{G} characterized by finitely many forbidden induced subgraphs is in logspace (See Lemma 2 in [17]). The proof of the next corollary follows from Lemma 2 [17] and Theorem 4.

Corollary 4. *Let the graph class \mathcal{G} be characterized by finitely many forbidden induced subgraphs $\mathcal{H} = \{H_1, H_2, \dots, H_l\}$ with $|V(H_i)| \leq d$ for all $1 \leq i \leq l$ where d is a constant. The graph isomorphism problem for graphs with bounded vertex deletion from \mathcal{G} is in L provided the colored graph isomorphism problem for graphs from \mathcal{G} is in L .*

Corollary 5. *The graph isomorphism problem is in L for following graph classes: (1) distance to interval graphs (2) distance to cographs.*

Proof. The proof of (1), follows from Theorem 4 and the logspace algorithm for colored GI for interval graphs (see [27]).

The proof of (2), follows from Corollary 4 and the logspace isomorphism algorithm for colored GI for cographs [25]. \square

6 Conclusion

In this paper we showed that graph isomorphism problem is in Para-TC^0 when parameterized by the vertex cover number of the input graphs. We also studied the parameterized complexity of graph isomorphism problem for the class of graphs \mathcal{G} characterized by finitely many forbidden induced subgraphs. We showed that graph isomorphism problem is in Para-AC^1 for the graphs in $\mathcal{G} + kv$ if there is an AC^1 algorithm for colored-GI for the graph class \mathcal{G} . From this result, we show that GI parameterized by the distance to cographs is in Para-AC^1 .

The following questions remain open. Can we get a parameterized logspace algorithm for GI parameterized by feedback vertex set number? Does the problem admit parameterized parallel algorithm? Elberfeld et al. [21] showed that GI is in logspace for graphs of bounded tree-width. In this paper, we showed that GI for some subclasses of bounded clique-width graphs is in L . It is an interesting open question to extend these results to bounded clique-width graphs.

References

1. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press, Cambridge (2009)
2. Babai, L.: Moderately exponential bound for graph isomorphism. In: Gécseg, F. (ed.) FCT 1981. LNCS, vol. 117, pp. 34–50. Springer, Heidelberg (1981). https://doi.org/10.1007/3-540-10854-8_4
3. Babai, L.: A Las Vegas-NC algorithm for isomorphism of graphs with bounded multiplicity of eigenvalues. In: 27th Annual Symposium on Foundations of Computer Science, pp. 303–312. IEEE (1986)
4. Babai, L.: Graph isomorphism in quasipolynomial time. In: 48th Annual ACM SIGACT Symposium on Theory of Computing, pp. 684–697. ACM (2016)
5. Bannach, M., Stockhusen, C., Tantau, T.: Fast parallel fixed-parameter algorithms via color coding. In: Parameterized and Exact Computation, p. 224 (2015)
6. Bodlaender, H.L.: Polynomial algorithms for graph isomorphism and chromatic index on partial k -trees. *J. Algorithms* **11**(4), 631–643 (1990)
7. Boppana, R.B., Hastad, J., Zachos, S.: Does co-NP have short interactive proofs? *Inform. Process. Lett.* **25**(2), 127–132 (1987)
8. Bouland, A., Dawar, A., Kopczyński, E.: On tractable parameterizations of graph isomorphism. In: Parameterized and Exact Computation, pp. 218–230 (2012)
9. Bulian, J., Dawar, A.: Graph isomorphism parameterized by elimination distance to bounded degree. *Algorithmica* **75**(2), 363–382 (2016)

10. Buss, J.F., Goldsmith, J.: Nondeterminism within P*. *SIAM J. Comput.* **22**(3), 560–572 (1993)
11. Cai, L.: Fixed-parameter tractability of graph modification problems for hereditary properties. *Inform. Process. Lett.* **58**(4), 171–176 (1996)
12. Cai, L., Chen, J., Downey, R.G., Fellows, M.R.: Advice classes of parameterized tractability. *Ann. Pure Appl. Logic* **84**(1), 119–138 (1997)
13. Cai, L., Chen, J., Downey, R.G., Fellows, M.R.: Advice classes of parameterized tractability-corrigendum (2017)
14. Chandoo, M.: Deciding circular-arc graph isomorphism in parameterized logspace. In: 33rd Symposium on Theoretical Aspects of Computer Science (2016)
15. Chen, J., Kanj, I.A., Xia, G.: Improved upper bounds for vertex cover. *Theoret. Comput. Sci.* **411**(40–42), 3736–3756 (2010)
16. Das, B., Enduri, M.K., Reddy, I.V.: Logspace and FPT algorithms for graph isomorphism for subclasses of bounded tree-width graphs. In: Rahman, M.S., Tomita, E. (eds.) WALCOM 2015. LNCS, vol. 8973, pp. 329–334. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15612-5_30
17. Das, B., Enduri, M.K., Reddy, I.V.: On the parallel parameterized complexity of the graph isomorphism problem. arXiv preprint [arXiv:1711.08885](https://arxiv.org/abs/1711.08885) (2017)
18. Datta, S., Limaye, N., Nimbhorkar, P., Thierauf, T., Wagner, F.: Planar graph isomorphism is in log-space. In: 24th Annual IEEE Conference on Computational Complexity, pp. 203–214 (2009)
19. Datta, S., Nimbhorkar, P., Thierauf, T., Wagner, F.: Graph isomorphism for $K_{3,3}$ -free and K_5 -free graphs is in log-space. In: LIPIcs-Leibniz International Proceedings in Informatics, vol. 4 (2009)
20. Downey, R.G., Fellows, M.R.: *Fundamentals of Parameterized Complexity*. Springer Science & Business Media, London (2013). <https://doi.org/10.1007/978-1-4471-5559-1>
21. Elberfeld, M., Schweitzer, P.: Canonizing graphs of bounded tree width in logspace. In: 33rd Symposium on Theoretical Aspects of Computer Science (2016)
22. Elberfeld, M., Stockhusen, C., Tantau, T.: On the space complexity of parameterized problems. In: *Parameterized and Exact Computation*, pp. 206–217 (2012)
23. Flum, J., Grohe, M.: Describing parameterized complexity classes. *Inf. Comput.* **187**(2), 291–319 (2003)
24. Ganian, R.: Improving vertex cover as a graph parameter. *Discrete Math. Theoret. Comput. Sci.* **17**(2), 77–100 (2015)
25. Grufien, B.: Capturing polynomial time using modular decomposition. In: 32nd Annual Symposium on Logic in Computer Science (LICS), pp. 1–12 (2017)
26. Hopcroft, J.E., Wong, J.K.: Linear time algorithm for isomorphism of planar graphs (preliminary report). In: *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, pp. 172–184. ACM (1974)
27. Köbler, J., Kuhnert, S., Laubner, B., Verbitsky, O.: Interval graphs: canonical representations in logspace. *SIAM J. Comput.* **40**(5), 1292–1315 (2011)
28. Kratsch, S., Schweitzer, P.: Isomorphism for graphs of bounded feedback vertex set number. In: Kaplan, H. (ed.) SWAT 2010. LNCS, vol. 6139, pp. 81–92. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13731-0_9
29. Lindell, S.: A logspace algorithm for tree canonization. In: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pp. 400–404. ACM (1992)
30. Lokshantov, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Fixed-parameter tractable canonization and isomorphism test for graphs of bounded treewidth. *SIAM J. Comput.* **46**(1), 161–189 (2017)

31. Luks, E.M.: Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comput. Syst. Sci.* **25**(1), 42–65 (1982)
32. Luks, E.M.: Parallel algorithms for permutation groups and graph isomorphism. In: 27th Symposium on Foundations of Computer Science, pp. 292–302 (1986)
33. Miller, G.: Isomorphism testing for graphs of bounded genus. In: Proceedings of 12th Annual ACM Symposium on Theory of Computing, pp. 225–235. ACM (1980)
34. Otachi, Y.: Isomorphism for graphs of bounded connected-path-distance-width. In: Chao, K.-M., Hsu, T., Lee, D.-T. (eds.) ISAAC 2012. LNCS, vol. 7676, pp. 455–464. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35261-4_48
35. Stockhusen, C.: On the space and circuit complexity of parameterized problems. Ph.D. thesis, Dissertation, Lübeck, Universität zu Lübeck, 2017 (2017)
36. Yamazaki, K., Bodlaender, H.L., de Fluiter, B., Thilikos, D.M.: Isomorphism for graphs of bounded distance width. *Algorithmica* **24**(2), 105–127 (1999)
37. Zemlyachenko, V., Konieko, N., Tyshkevich, R.: Graph isomorphism problem (Russian). In: The Theory of Computation I, Notes Sci. Sem. LOMI 118 (1982)

Author Index

- Agbor, Bateh Mathias 20
Ahmed, Abu Reyan 156
Ahn, Hee-Kap 44, 56
Ahn, Taehoon 44, 56
Aljohani, Aisha 169
Azim, Md. Aashikur Rahman 183
- Bae, Sang Won 44
Baig, Mirza Galib Anwarul Husain 68
- Cho, Junhee 132
Choi, Jongmin 44, 56
Choudhari, Jayesh 228
- Das, Bireswar 252
Demange, Marc 144
- Enduri, Murali Krishna 252
- Fujita, Takahiro 195
Fujito, Toshihiro 32
- Gu, Hanyu 119
- Hatano, Kohei 195
Hayashi, Yu-ichi 20
Hiraishi, Hidefumi 216
- Imai, Hiroshi 216
Itoh, Toshiya 106
- Kabir, Mohimenul 183
Kashyop, Manas Jyoti 80
Kesh, Deepanjan 68
Kim, Mincheol 44, 56
Kimura, Kei 32
Kiyomi, Masashi 8
Kobourov, Stephen 156
- Masuda, Shingo 20
Memar, Julia 119
Miyano, Eiji 240
Mizuki, Takaaki 20
Mizuno, Yuki 32
- Nagayama, Tsunehiko 80
Nakano, Shin-ichi 1
- Ogasawara, Tomoaki 216
Oh, Eunjin 44, 56
Olsen, Martin 144
- Park, Sewon 132
Poudel, Pavan 169
- Qin, Tong 93
- Rahman, Atif 207
Rahman, M. Sohel 183, 207
Rahman, Md. Saidur 156
Reddy, I. Vinod 228, 252
Ruangwises, Suthee 106
- Sadakane, Kunihiko 80
Saitoh, Toshiki 8, 240
Sasaki, Tatsuya 20
Sayeed, Suri Dipannita 207
Sharma, Gokarna 169
Shin, Chan-Su 44
Shiroshita, Shinya 216
Sone, Hideaki 20
- Takimoto, Eiji 195
- Uehara, Ryuhei 8, 240
- van der Zanden, Tom C. 240
- Watanabe, Osamu 93
- Yagita, Tsuyoshi 240
Yamazaki, Kazuaki 8
Yoon, Sang Duk 44
- Ziegler, Martin 132
Zinder, Yakov 119