

Ding-Zhu Du  
Guochuan Zhang (Eds.)

LNCS 7936

# Computing and Combinatorics

19th International Conference, COCOON 2013  
Hangzhou, China, June 2013  
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, UK*

Takeo Kanade

*Carnegie Mellon University, Pittsburgh, PA, USA*

Josef Kittler

*University of Surrey, Guildford, UK*

Jon M. Kleinberg

*Cornell University, Ithaca, NY, USA*

Alfred Kobsa

*University of California, Irvine, CA, USA*

Friedemann Mattern

*ETH Zurich, Switzerland*

John C. Mitchell

*Stanford University, CA, USA*

Moni Naor

*Weizmann Institute of Science, Rehovot, Israel*

Oscar Nierstrasz

*University of Bern, Switzerland*

C. Pandu Rangan

*Indian Institute of Technology, Madras, India*

Bernhard Steffen

*TU Dortmund University, Germany*

Madhu Sudan

*Microsoft Research, Cambridge, MA, USA*

Demetri Terzopoulos

*University of California, Los Angeles, CA, USA*

Doug Tygar

*University of California, Berkeley, CA, USA*

Gerhard Weikum

*Max Planck Institute for Informatics, Saarbruecken, Germany*

Ding-Zhu Du Guochuan Zhang (Eds.)

# Computing and Combinatorics

19th International Conference, COCOON 2013  
Hangzhou, China, June 21-23, 2013  
Proceedings

Volume Editors

Ding-Zhu Du  
University of Texas at Dallas  
Department of Computer Science  
800W, Campbell Road, Richardson, TX 75080, USA  
E-mail: dzdu@utdallas.edu

Guochuan Zhang  
Zhejiang University  
College of Computer Science and Technology  
38, Zheda Road, Hangzhou 310027, China  
E-mail: zgc@zju.edu.cn

ISSN 0302-9743

e-ISSN 1611-3349

ISBN 978-3-642-38767-8

e-ISBN 978-3-642-38768-5

DOI 10.1007/978-3-642-38768-5

Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013939382

CR Subject Classification (1998): F.2, C.2, G.2, F.1, E.1, I.3.5

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

© Springer-Verlag Berlin Heidelberg 2013

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

The use of general descriptive names, registered names, trademarks, 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.

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

Printed on acid-free paper

Springer is part of Springer Science+Business Media ([www.springer.com](http://www.springer.com))

# Preface

The 19th International Computing and Combinatorics Conference (COCOON 2013) took place in Hangzhou, China, during June 21–23, 2013. COCOON 2013 provided a forum for researchers working in the area of theoretical computer science and combinatorics.

Submissions to the conference this year were conducted electronically with the Spring Online Conference Service (OCS). A totally of 120 papers were submitted from more than 15 countries and regions. Through an evaluation by the international Program Committee, 56 papers were accepted for the main conference of COCOON and 8 short papers were accepted for a workshop on discrete algorithms co-organized by Ding-Zhu Du, Xiaodong Hu, and Guochuan Zhang. Some of these accepted papers will be selected for publication in a special issue of *Algorithmica*, a special issue of *Theoretical Computer Science*, and a special issue of *Journal of Combinatorial Optimization*. It is expected that the journal version papers will appear in a more complete form.

Co-located with COCOON 2013, there was a workshop on computational social networks (CSoNet 2013) held on June 22, 2013. An independent Program Committee chaired by My Thai and Arun Sen reviewed 25 submissions which 12 papers were selected for presentation at the workshop. We appreciate the work by the CSoNet Program Committee that helped enrich the conference topics.

We would like to thank the two eminent keynote speakers, Susanne Albers of Humboldt University and Robert Kleinberg of Cornell University for their contribution to the conference and the proceedings.

We wish to thank all authors who submitted their papers to the conference. We are grateful to all members of the Program Committee and the external referees for their excellent work within the time constraints. We wish to thank all members of the Organizing Committee for their assistance and contribution in the advertising, Web management, and local arrangements etc., which contributed success of the conference.

Finally, we wish to thank the conference sponsors for their support.

June 2013

Ding-Zhu Du  
Guochuan Zhang

# Organization

## Program Committee

Hee-Kap Ahn	Pohang University of Science and Technology, Korea
Yossi Azar	Tel Aviv University, Israel
Leizhen Cai	The Chinese University of Hong Kong, SAR China
Zhipeng Cai	Georgia State University, USA
Wei Chen	Microsoft Research Asia
Zhixiang Chen	University of Texas-Pan American, USA
Altannar Chinchuluun	National University of Mongolia
Janos Csirik	University of Szeged, Hungary
Ding-Zhu Du	University of Texas at Dallas, Co-chair, USA
Xiaofeng Gao	Shanghai Jiaotong University, China
Qianping Gu	Simon Fraser University, Canada
Xiaodong Hu	Chinese Academy of Sciences, China
Klaus Jansen	University of Kiel, Germany
Iyad Kanj	DePaul University, USA
Ming-Yang Kao	Northwestern University, USA
Piotr Krysta	University of Liverpool, UK
Jian Li	Tsinghua University, China
Chi-Jen Lu	Academia Sinica, Taiwan
Julian Mestre	University of Sydney, Australia
Mitsunori Ogihara	University of Miami, USA
Jiri Sgall	Charles University, Czech Republic
He Sun	Max Planck Institute, Germany
Maxim Sviridenko	University of Warwick, UK
My Thai	University of Florida, USA
Takeshi Tokuyama	Tohoku University, Japan
Marc Uetz	University of Twente, The Netherlands
Jianping Yin	National University of Defense Technology, China
Alex Zelikovsky	Georgia State University, USA
Guochuan Zhang	Zhejiang University, Co-chair, China

## CSoNet Program Committee

Ginestra Bianconi	Northeastern University, USA
Sujogya Banerjee	Arizona State University, USA
Guanling Chen	University of Massachusetts Lowell, USA

## VIII Organization

Xiaohua Jia	City University of Hong Kong, SAR China
Timothy Killingback	University of Massachusetts Boston, USA
Yingshu Li	Georgia State University, USA
Panos Pardalos	University of Florida, USA
Andrea Richa	Arizona State University, USA
Arun Sen	Arizona State University, Co-chair, USA
My Thai	University of Florida, Co-chair, USA
Steve Uhlig	Queen Mary, University of London, UK
Jie Wang	University of Massachusetts Lowell, USA
Feng Wang	Arizona State University, USA
Yajun Wang	Microsoft Research Asia
Guoliang Xue	Arizona State University, USA
Zhi-Li Zhang	University of Minnesota, USA

## Organizing Committee

Lin Chen	Zhejiang University
Lili Mei	Zhejiang University
Zhiyi Tan	Zhejiang University
Deshi Ye	Zhejiang University, Co-chair
Guochuan Zhang	Zhejiang University, Co-chair

## Sponsors

National Natural Science Foundation of China(11271325)  
College of Computer Science, Zhejiang University  
The Mathematical Programming Branch of OR Society of China

## External Reviewers

Antoniadis, Antonios	Chen, Po-An	Gaspers, Serge
Asano, Tetsuo	Chen, Danny	Goel, Gagan
Bansal, Nikhil	Chen, Yu-Fang	Golin, Mordecai
Bei, Xiaohui	Cheng, Siu-Wing	Golovach, Petr
Bein, Wolfgang	Chrobak, Marek	Gregor, Petr
Berebrink, petra	Cohen, Ilan	Grigoriev, Alexander
Bernstein, Aaron	Ding, Ling	Groß, Martin
Bley, Andreas	Dosa, Gyorgy	Guo, Zeyu
Bodirsky, Manuel	Duan, Ran	Guttmann, Tony
Buchbinder, Niv	Epstein, Leah	Górecki, Paweł
Byrka, Jarek	Fiat, Amos	Huang, Zhiyi
Cai, Xiaojuan	Fotakis, Dimitris	Hwang, Yoonho
Chang, Huilan	Frati, Fabrizio	Im, Sungjin
Chen, Yijia	Garing, Martin	Ito, Takehiro

Jun, Li	Lübbecke, Marco	Shin, Chan-Su
Jung, Hyunwoo	Ma, Bin	Shioura, Akiyoshi
Kaufmann, Michael	Mahdian, Mohammad	Simon, Sunil
Khandekar, Rohit	Manthey, Bodo	Son, Wanbin
Kim, Donghyun	Megow, Nicole	Starnberger, Martin
Kim, Hyo-Sil	Mnich, Matthias	van Stee, Rob
Kim, Sang-Sub	Nasre, Meghana	Syrkanis, Vasilis
Klein, Kim-Manuel	Nguyen, Kim Thang	Telelis, Orestis
Klimm, Max	Nonner, Tim	Tian, Cong
Knop, Dušan	Okamoto, Yoshio	Ting, Hingfung
Kononov, Alexander	Otachi, Yota	Tong, Weitian
Kraft, Stefan	Papamichail, Dimitris	Toth, Csaba
Krysta, Piotr	Park, Dongwoo	Uehara, Ryuhei
Li, Guoqiang	Paulusma, Daniel	Wang, Yajun
Lam, Tak-Wah	Perkovic, Ljubomir	Wang, Wei
Land, Kati	Post, Gerhard	Ward, Justin
Land, Felix	Pountourakis, Emmanouil	Wong, Prudence
Le, Van Bang	Pruhs, Kirk	Wu, Chenye
Li, Shi	Rahman, Md Saidur	Xia, Mingji
Li, Xianyue	Rogers, John	Xia, Ge
Liang, Hongyu	Sach, Benjamin	Yang, Yongtian
Liao, Kewen	Samal, Robert	Yoon, Sang Duk
Liu, Yang	Sanita, Laura	Zhang, Jie
Liu, Zhixin	Sauerwald, Thomas	Zhang, Zhao
Long, Huan	Schaefer, Marcus	Zhang, Chihao
Lu, Pinyan	Shi, Yan	Zhong, Jiaofei
Lucarelli, Giorgio		Zhu, Yuqing

# Table of Contents

## Keynote

Recent Results for Online Makespan Minimization . . . . .	1
<i>Susanne Albers</i>	
Optimal Stopping Meets Combinatorial Optimization . . . . .	4
<i>Robert Kleinberg</i>	

## Game Theory

New Bounds for the Balloon Popping Problem . . . . .	5
<i>Davide Bilò and Vittorio Bilò</i>	
On the Sequential Price of Anarchy of Isolation Games . . . . .	17
<i>Anna Angelucci, Vittorio Bilò, Michele Flammini, and Luca Moscardelli</i>	
Social Exchange Networks with Distant Bargaining . . . . .	29
<i>Konstantinos Georgiou, George Karakostas, Jochen Könemann, and Zuzanna Stamirowska</i>	
The 1/4-Core of the Uniform Bin Packing Game Is Nonempty . . . . .	41
<i>Walter Kern and Xian Qiu</i>	

## Randomized Algorithms

On the Advice Complexity of the Online $L(2, 1)$ -Coloring Problem on Paths and Cycles . . . . .	53
<i>Maria Paola Bianchi, Hans-Joachim Böckenhauer, Juraj Hromkovič, Sacha Krug, and Björn Steffen</i>	
On Randomized Fictitious Play for Approximating Saddle Points over Convex Sets . . . . .	65
<i>Khaled Elbassioni, Kazuhisa Makino, Kurt Mehlhorn, and Fahimeh Ramezani</i>	
A Fast Algorithm for Data Collection along a Fixed Track . . . . .	77
<i>Otfried Cheong, Radwa El Shawi, and Joachim Gudmundsson</i>	
Random Methods for Parameterized Problems . . . . .	89
<i>Qilong Feng, Jianxin Wang, Shaohua Li, and Jianer Chen</i>	

## Scheduling Algorithms

DVS Scheduling in a Line or a Star Network of Processors .....	101
<i>Zongxu Mu and Minming Li</i>	
Online Algorithms for Batch Machines Scheduling with Delivery Times .....	114
<i>Peihai Liu and Xiwen Lu</i>	
How to Schedule the Marketing of Products with Negative Externalities .....	122
<i>Zhigang Cao, Xujin Chen, and Changjun Wang</i>	
From Preemptive to Non-preemptive Speed-Scaling Scheduling .....	134
<i>Evripidis Bampis, Alexander Kononov, Dimitrios Letsios, Giorgio Lucarelli, and Ioannis Nemparis</i>	

## Computational Theory

Selection from Read-Only Memory with Limited Workspace .....	147
<i>Amr Elmasry, Daniel Dahl Juhl, Jyrki Katajainen, and Srinivasa Rao Satti</i>	
Determinization of Büchi Automata as Partitioned Automata .....	158
<i>Cong Tian, Zhenhua Duan, and Mengfei Yang</i>	
On Linear-Size Pseudorandom Generators and Hardcore Functions .....	169
<i>Joshua Baron, Yuval Ishai, and Rafail Ostrovsky</i>	
A Fast Algorithm Finding the Shortest Reset Words.....	182
<i>Andrzej Kisielewicz, Jakub Kowalski, and Marek Szykuła</i>	

## Computational Geometry

The Discrete Voronoi Game in a Simple Polygon .....	197
<i>Aritra Banik, Sandip Das, Anil Maheshwari, and Michiel Smid</i>	
Facets for Art Gallery Problems .....	208
<i>Sándor P. Fekete, Stephan Friedrichs, Alexander Kröller, and Christiane Schmidt</i>	
Hitting and Piercing Rectangles Induced by a Point Set .....	221
<i>Ninad Rajgopal, Pradeesha Ashok, Sathish Govindarajan, Abhijit Khopkar, and Neeldhara Misra</i>	

Realistic Roofs over a Rectilinear Polygon Revisited . . . . .	233
<i>Jessica Sherette and Sang Duk Yoon</i>	

## Graph Algorithms I

Parametric Power Supply Networks (Extended Abstract) . . . . .	245
<i>Shiho Morishita and Takao Nishizeki</i>	
Approximating the Minimum Independent Dominating Set in Perturbed Graphs . . . . .	257
<i>Weitian Tong, Randy Goebel, and Guohui Lin</i>	
A Linear-Time Algorithm for the Minimum Degree Hypergraph Problem with the Consecutive Ones Property . . . . .	268
<i>Chih-Hsuan Li, Jhih-Hong Ye, and Biing-Feng Wang</i>	

On the Conjunctive Capacity of Graphs . . . . .	280
<i>Miroslav Chlebík and Janka Chlebíková</i>	

## Approximation Algorithms

Improved Approximation Algorithms for the Facility Location Problems with Linear/submodular Penalty . . . . .	292
<i>Yu Li, Donglei Du, Naihua Xiu, and Dachuan Xu</i>	

An Improved Semidefinite Programming Hierarchies Rounding Approximation Algorithm for Maximum Graph Bisection Problems . . . . .	304
<i>Chenchen Wu, Donglei Du, and Dachuan Xu</i>	

Improved Local Search for Universal Facility Location . . . . .	316
<i>Eric Angel, Nguyen Kim Thang, and Damien Regnault</i>	

Improved Approximation Algorithms for Computing $k$ Disjoint Paths Subject to Two Constraints . . . . .	325
<i>Longkun Guo, Hong Shen, and Kewen Liao</i>	

## Graph Algorithms II

The $k$ -Separator Problem . . . . .	337
<i>Walid Ben-Ameur, Mohamed-Ahmed Mohamed-Sidi, and José Neto</i>	

On the Treewidth of Dynamic Graphs . . . . .	349
<i>Bernard Mans and Luke Mathieson</i>	

Square-Orthogonal Drawing with Few Bends per Edge . . . . .	361
<i>Yu-An Lin and Sheung-Hung Poon</i>	

Covering Tree with Stars.....	373
<i>Jan Baumbach, Jiong Guo, and Rashid Ibragimov</i>	

## Computational Biology

A Polynomial Time Approximation Scheme for the Closest Shared Center Problem.....	385
<i>Weidong Li, Lusheng Wang, and Wenjuan Cui</i>	

An Improved Approximation Algorithm for Scaffold Filling to Maximize the Common Adjacencies .....	397
---	-----

*Nan Liu, Haitao Jiang, Daming Zhu, and Binhai Zhu*

An Efficient Algorithm for One-Sided Block Ordering Problem with Block-Interchange Distance .....	409
<i>Kun-Tze Chen, Chi-Long Li, Chung-Han Yang, and Chin Lung Lu</i>	

A Combinatorial Approach for Multiple RNA Interaction: Formulations, Approximations, and Heuristics .....	421
<i>Syed Ali Ahmed, Saad Mneimneh, and Nancy L. Greenbaum</i>	

## Graph Algorithms III

Maximum Balanced Subgraph Problem Parameterized above Lower Bound .....	434
<i>Robert Crowston, Gregory Gutin, Mark Jones, and Gabriele Muciaccia</i>	

A Toolbox for Provably Optimal Multistage Strict Group Testing Strategies .....	446
<i>Peter Damaschke and Azam Sheikh Muhammad</i>	

A Linear Edge Kernel for Two-Layer Crossing Minimization .....	458
<i>Yasuaki Kobayashi, Hirokazu Maruta, Yusuke Nakae, and Hisao Tamaki</i>	

A Linear-Time Algorithm for Computing the Prime Decomposition of a Directed Graph with Regard to the Cartesian Product .....	469
<i>Christophe Crespelle, Eric Thierry, and Thomas Lambert</i>	

## Online Algorithms

Metrical Service Systems with Multiple Servers .....	481
<i>Ashish Chiplunkar and Sundar Vishwanathan</i>	

The String Guessing Problem as a Method to Prove Lower Bounds on the Advice Complexity (Extended Abstract) .....	493
<i>Hans-Joachim Böckenhauer, Juraj Hromkovič, Dennis Komm, Sacha Krug, Jasmin Smula, and Andreas Srock</i>	
Online Algorithms for 1-Space Bounded 2-Dimensional Bin Packing and Square Packing .....	506
<i>Yong Zhang, Francis Y.L. Chin, Hing-Fung Ting, Xin Han, Chung Keung Poon, Yung H. Tsin, and Deshi Ye</i>	
Improved Lower Bounds for the Online Bin Packing Problem with Cardinality Constraints .....	518
<i>Hiroshi Fujiiwara and Koji Kobayashi</i>	

## Parameterized Algorithms

Parameterized Complexity of Flood-Filling Games on Trees .....	531
<i>Uéverton dos Santos Souza, Fábio Protti, and Maise Dantas da Silva</i>	
Parameterized Approximability of Maximizing the Spread of Influence in Networks .....	543
<i>Cristina Bazgan, Morgan Chopin, André Nichterlein, and Florian Sikora</i>	
An Effective Branching Strategy for Some Parameterized Edge Modification Problems with Multiple Forbidden Induced Subgraphs ....	555
<i>Yunlong Liu, Jianxin Wang, Chao Xu, Jiong Guo, and Jianer Chen</i>	
Parameterized Algorithms for Maximum Agreement Forest on Multiple Trees .....	567
<i>Feng Shi, Jianer Chen, Qilong Feng, and Jianxin Wang</i>	

## Computational Complexity

Small $H$ -Coloring Problems for Bounded Degree Digraphs.....	579
<i>Pavol Hell and Auroshish Mishra</i>	
Bounded Model Checking for Propositional Projection Temporal Logic .....	591
<i>Zhenhua Duan, Cong Tian, Mengfei Yang, and Jia He</i>	
Packing Cubes into a Cube Is NP-Hard in the Strong Sense .....	603
<i>Yiping Lu, Danny Z. Chen, and Jianzhong Cha</i>	
On the Complexity of Solving or Approximating Convex Recoloring Problems .....	614
<i>Manoel B. Campêlo, Cristiana G. Huiban, Rudini M. Sampaio, and Yoshiko Wakabayashi</i>	

## Algorithms

2-connecting Outerplanar Graphs without Blowing Up the Pathwidth . . . . .	626
<i>Jasine Babu, Manu Basavaraju, Sunil Chandran Leela, and Deepak Rajendraprasad</i>	
How to Catch $L_2$ -Heavy-Hitters on Sliding Windows . . . . .	638
<i>Vladimir Braverman, Ran Gelles, and Rafail Ostrovsky</i>	
Time/Memory/Data Tradeoffs for Variants of the RSA Problem . . . . .	651
<i>Pierre-Alain Fouque, Damien Vergnaud, and Jean-Christophe Zapalowicz</i>	
An Improved Algorithm for Extraction of Exact Boundaries and Boundaries Inclusion Relationship . . . . .	663
<i>Tao Hu, Xianyi Ren, and Jihong Zhang</i>	

## Workshop I

Straight-Line Monotone Grid Drawings of Series-Parallel Graphs . . . . .	672
<i>Md. Iqbal Hossain and Md. Saidur Rahman</i>	
Combination of Two-Machine Flow Shop Scheduling and Shortest Path Problems . . . . .	680
<i>Kameng Nip and Zhenbo Wang</i>	
The Program Download Problem: Complexity and Algorithms . . . . .	688
<i>Chao Peng, Jie Zhou, Binhai Zhu, and Hong Zhu</i>	
Finding Theorems in NBG Set Theory by Automated Forward Deduction Based on Strong Relevant Logic . . . . .	697
<i>Hongbiao Gao, Kai Shi, Yuichi Goto, and Jingde Cheng</i>	

## Workshop II

Perturbation Analysis of Maximum-Weighted Bipartite Matchings with Low Rank Data . . . . .	705
<i>Xingwu Liu and Shang-Hua Teng</i>	
Sublinear Time Approximate Sum via Uniform Random Sampling . . . . .	713
<i>Bin Fu, Wenfeng Li, and Zhiyong Peng</i>	
Tractable Connected Domination for Restricted Bipartite Graphs . . . . .	721
<i>Zhao Lu, Tian Liu, and Ke Xu</i>	

- On the Minimum Caterpillar Problem in Digraphs ..... 729  
*Taku Okada, Akira Suzuki, Takehiro Ito, and Xiao Zhou*

## CSoNet I

- A New Model for Product Adoption over Social Networks ..... 737  
*Lidan Fan, Zaixin Lu, Weili Wu, Yuanjun Bi, and Ailian Wang*
- Generating Uncertain Networks Based on Historical Network Snapshots ..... 747  
*Meng Han, Mingyuan Yan, Jinbao Li, Shouling Ji, and Yingshu Li*
- A Short-Term Prediction Model of Topic Popularity on Microblogs ..... 759  
*Juanjuan Zhao, Weili Wu, Xiaolong Zhang, Yan Qiang, Tao Liu, and Lidong Wu*
- Social Network Path Analysis Based on HBase ..... 770  
*Yan Qiang, Junzuo Lu, Weili Wu, Juanjuan Zhao, Xiaolong Zhang, Yue Li, and Lidong Wu*

## CSoNet II

- Community Expansion Model Based on Charged System Theory ..... 780  
*Yuanjun Bi, Weili Wu, Ailian Wang, and Lidan Fan*
- Centrality and Spectral Radius in Dynamic Communication Networks ..... 791  
*Danica Vukadinović Greetham, Zhivko Stoyanov, and Peter Grindrod*
- Finding Network Communities Using Random Walkers with Improved Accuracy ..... 801  
*You Li, Jie Wang, Benyuan Liu, and Qilian Liang*
- Homophilic and Communities Detection among a Subset of Blogfa Persian Weblogs: Computer and Internet Category ..... 811  
*Adib Rastegarnia, Meysam Mohajer, and Vahid Solouk*

## CSoNet III

- Neighborhood-Based Dynamic Community Detection with Graph Transform for 0-1 Observed Networks ..... 821  
*Li Wang, Yuanjun Bi, Weili Wu, Biao Lian, and Wen Xu*
- Effects of Inoculation Based on Structural Centrality on Rumor Dynamics in Social Networks ..... 831  
*Anurag Singh, Rahul Kumar, and Yatindra Nath Singh*

XVIII Table of Contents

A Dominating Set Based Approach to Identify Effective Leader Group of Social Network .....	841
<i>Donghyun Kim, Deying Li, Omid Asgari, Yingshu Li, and     Alade O. Tokuta</i>	
Using Network Sciences to Evaluate the Brazilian Airline Network .....	849
<i>Douglas Oliveira, Marco Carvalho, and Ronaldo Menezes</i>	
<b>Author Index .....</b>	<b>859</b>

# Recent Results for Online Makespan Minimization

## (Extended Abstract)

Susanne Albers

Department of Computer Science  
Humboldt-Universität zu Berlin  
[albers@informatik.hu-berlin.de](mailto:albers@informatik.hu-berlin.de)

**Overview:** We study a classical scheduling problem that has been investigated for more than forty years. Consider a sequence of jobs  $\sigma = J_1, \dots, J_n$  that has to be scheduled on  $m$  identical parallel machines. Each job  $J_t$  has an individual processing time  $p_t$ ,  $1 \leq t \leq n$ . Preemption of jobs is not allowed. The goal is to minimize the makespan, i.e. the maximum completion time of any job in the constructed schedule. In the offline variant of the problem all jobs of  $\sigma$  are known in advance. In the online variant the jobs arrive one by one. Each incoming job  $J_t$  has to be assigned immediately to one of the machines without knowledge of any future jobs  $J_{t'}, t' > t$ .

Already in the 1960s Graham [7] presented the famous *List* scheduling algorithm, which schedules each job of  $\sigma$  on a machine that currently has the smallest load. *List* can be used in the offline as well as the online setting. Graham proved that the strategy is  $(2 - 1/m)$ -competitive, i.e. for any  $\sigma$  the makespan of *List*'s schedule is at most  $2 - 1/m$  times the makespan of an optimal schedule. In the 1980s Hochbaum and Shmoys [8] developed a famous polynomial time approximation scheme for the offline problem. Research over the past two decades has focused on the online problem. The best deterministic online strategy currently known achieves a competitive ratio of about 1.92, see [6]. Moreover, no deterministic online algorithm can attain a competitiveness smaller than 1.88, cf. [10].

During the last years online makespan minimization has been explored assuming that an online algorithm is given additional power or extra information while processing a job sequence  $\sigma$ . It turns out that usually significantly improved competitive ratios can be achieved. We survey these recent advances. The considered forms of resource augmentation are generally well motivated from a practical point of view.

**Job migration:** Assume that at any time an online algorithm may perform reassignments, i.e. jobs already assigned to machines may be removed and transferred to other machines. Job migration is a well-known and widely used technique to balance load in parallel and distributed systems. Sanders et al. [11] study the setting that after the arrival of each job  $J_t$ , jobs up to a processing volume of  $\beta p_t$  may be migrated, where  $\beta$  is a constant. For  $\beta = 4/3$ , they present a 1.5-competitive algorithm. They also devise a  $(1 + \epsilon)$ -competitive

algorithm, for any  $\epsilon > 0$ , where  $\beta$  depends exponentially on  $1/\epsilon$ . Albers and Hellwig [1] investigate the scenario where an online algorithm may migrate a limited number of jobs; this number does not depend on the job sequence. They give an algorithm that, for any  $m \geq 2$ , is  $\alpha_m$ -competitive, where  $\lim_{m \rightarrow \infty} \alpha_m = W_{-1}(-1/e^2)/(1 + W_{-1}(-1/e^2)) \approx 1.4659$ . Here  $W_{-1}$  is the lower branch of the Lambert  $W$  function. The total number of migrations is at most  $7m$ , for  $m \geq 11$ . The competitiveness of  $\alpha_m$  is best possible: No deterministic online algorithm that uses  $o(n)$  migrations can achieve a competitive ratio smaller than  $\alpha_m$ .

**Availability of a reordering buffer:** In this setting an online algorithm has a buffer of limited size that may be used to partially reorder the job sequence. Whenever a job arrives, it is inserted into the buffer; then one job of the buffer is removed and assigned in the current schedule. For  $m = 2$  machines, Zhang [12] and Kellerer et al. [9] give  $4/3$ -competitive algorithms. Englert et al. [5] explore the setting for general  $m$ . They develop an algorithm that, using a buffer of size  $O(m)$ , is  $\alpha_m$ -competitive, where again  $\lim_{m \rightarrow \infty} \alpha_m = W_{-1}(-1/e^2)/(1 + W_{-1}(-1/e^2)) \approx 1.4659$ . Furthermore they prove that if an online algorithm achieves a competitiveness smaller than  $\alpha_m$ , then the buffer size must depend on  $\sigma$ . The paper by Englert et al. [5] also consideres makespan minimization with a reordering buffer on uniformly related machines.

**Information on total processing time or optimum makespan:** First assume that an online algorithm knows the sum  $\sum_{t=1}^n p_t$  of the jobs' processing times. The access to such a piece of information can be justified as follows. In a parallel server system there usually exist fairly accurate estimates on the workload that arrives over a given time horizon. Furthermore, in a shop floor a scheduler typically accepts orders (tasks) of a targeted volume for a given time period, say a day or a week. For  $m = 2$  machines, Kellerer et al. [9] present a  $4/3$ -competitive algorithm. For a general number  $m$  of machines, Cheng et al. [4] propose a  $1.6$ -competitive algorithm. Albers and Hellwig [2] prove that no deterministic online algorithm can attain a competitiveness smaller than  $1.585$ . In a stronger scenario an online algorithm even knows the value of the optimum makespan, for the given  $\sigma$ . This framework can also be viewed as a bin stretching problem, see [3]. Obviously, the algorithm by Cheng et al. [4] is also  $1.6$ -competitive in this setting. Azar and Regev [3] show that no online algorithm can be better than  $4/3$ -competitive.

## References

1. Albers, S., Hellwig, M.: On the value of job migration in online makespan minimization. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 84–95. Springer, Heidelberg (2012)
2. Albers, S., Hellwig, M.: Semi-online scheduling revisited. Theoretical Computer Science 443, 1–9 (2012)
3. Azar, Y., Regev, O.: On-line bin-stretching. Theoretical Computer Science 268, 17–41 (2001)
4. Cheng, T.C.E., Kellerer, H., Kotov, V.: Semi-on-line multiprocessor scheduling with given total processing time. Theoretical Computer Science 337, 134–146 (2005)

5. Englert, M., Özmen, D., Westermann, M.: The power of reordering for online minimum makespan scheduling. In: Proc. 49th Annual IEEE Symposium on Foundations of Computer Science, pp. 603–612 (2008)
6. Fleischer, R., Wahl, M.: Online scheduling revisited. *Journal of Scheduling* 3, 343–353 (2000)
7. Graham, R.L.: Bounds for certain multi-processing anomalies. *Bell System Technical Journal* 45, 1563–1581 (1966)
8. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM* 34, 144–162 (1987)
9. Kellerer, H., Kotov, V., Speranza, M.G., Tuza, Z.: Semi on-line algorithms for the partition problem. *Operations Research Letters* 21, 235–242 (1997)
10. Rudin III, J.F.: Improved bounds for the on-line scheduling problem. Ph.D. Thesis. The University of Texas at Dallas (May 2001)
11. Sanders, P., Sivadasan, N., Skutella, M.: Online scheduling with bounded migration. *Mathematics of Operations Research* 34(2), 481–498 (2009)
12. Zhang, G.: A simple semi on-line algorithm for  $P2//C_{\max}$  with a buffer. *Information Processing Letters* 61, 145–148 (1997)

# Optimal Stopping Meets Combinatorial Optimization

Robert Kleinberg

Cornell University  
Ithaca, NY 14850, USA  
[robert.kleinberg@cornell.edu](mailto:robert.kleinberg@cornell.edu)  
<http://www.cs.cornell.edu/~rdk>

**Abstract.** Optimal stopping theory considers the design of online algorithms for stopping a random sequence subject to an optimization criterion. For example, the famous secretary problem asks to identify a stopping rule that maximizes the probability of selecting the maximum element in a sequence presented in uniformly random order. In a similar vein, the prophet inequality of Krengel, Sucheston, and Garling establishes the existence of an online algorithm for selecting one element from a sequence of independent random numbers, such that the expected value of the chosen element is at least half the expectation of the maximum.

A rich set of problems emerges when one combines these models with notions from combinatorial optimization by allowing the algorithm to select multiple elements from the sequence, subject to a combinatorial feasibility constraint on the set selected. A sequence of results during the past ten years have contributed greatly to our understanding of these problems. I will survey some of these developments and their applications to topics in algorithmic game theory.

# New Bounds for the Balloon Popping Problem\*

Davide Bilò<sup>1</sup> and Vittorio Bilò<sup>2</sup>

<sup>1</sup> Dipartimento di Teorie e Ricerche dei Sistemi Culturali, University of Sassari  
Piazza Conte di Moriana, 8, 07100 Sassari, Italy  
[davide.bilo@uniss.it](mailto:davide.bilo@uniss.it)

<sup>2</sup> Department of Mathematics and Physics “Ennio De Giorgi”, University of Salento  
Provinciale Lecce-Arnesano, P.O. Box 193, 73100 Lecce, Italy  
[vittorio.bilo@unisalento.it](mailto:vittorio.bilo@unisalento.it)

**Abstract.** We reconsider the balloon popping problem, an intriguing combinatorial problem introduced in order to bound the competitiveness of ascending auctions with anonymous bidders with respect to the best fixed-price scheme. Previous works show that the optimal solution for this problem is in the range [1.6595, 2]. We give a new lower bound of 1.68 and design an  $O(n^5)$  algorithm for computing upper bounds as a function of the number of bidders  $n$ . Our algorithm provides an experimental evidence that the correct upper bound is smaller than 2, thus disproving a currently believed conjecture, and can be used to test the validity of a new conjecture we propose, according to which the upper bound would decrease to  $\pi^2/6 + 1/4 \approx 1.8949$ .

## 1 Introduction

In digital goods auctions, also known as unlimited supply auctions [2,5], an auctioneer sells a collection of identical items to unit-demand bidders. As usual in this setting, when getting the item, a bidder experiences a certain utility which is unknown to the auctioneer. Given this restriction, the auctioneer wants to design a mechanism, i.e., a set of auction rules, so as to raise as much revenue as possible from the bidders. The competitiveness of a certain mechanism is usually measured with respect to the maximum revenue that can be raised by an omniscient auctioneer who is restricted to offer the goods at the same price to all bidders (*fixed-price scheme*) [1,2,3,4]. One can ask whether this is a reasonable restriction, and whether without this restriction, the auctioneer can achieve a considerably higher revenue.

Immorlica *et al.* [6] provided an answer to this question for the case of *ascending auctions* with *anonymous bidders*. More precisely, they consider the scenario in which *i*) the auctioneer knows the set of the bidders’ utilities, but is unable to determine which bidder has which utility, and *ii*) during the auction, the auctioneer can only rise the price offered to a bidder. In particular, Immorlica *et*

---

\* This work was partially supported by the PRIN 2010–2011 research project ARS TechnoMedia: “Algorithmics for Social Technological Networks” funded by the Italian Ministry of University.

*al.* [6] show that, under conditions *i)* and *ii)*, no mechanism can raise a revenue of more than a constant times the one raised by the best fixed-price scheme.

Such a result is achieved by introducing and studying the *balloon popping problem* which is defined as follows.

*We are given  $n$  undistinguishable balloons of capacities  $1, 1/2, 1/3, \dots, 1/n$  and we are asked to blow them so as to maximize the total volume of air inflated in the balloons knowing that a balloon blown up beyond its capacity pops (and reveals its capacity) thus giving no contribution to the total volume. What is the best blowing strategy and what total volume is achievable?*

A blowing strategy is said to be *offline* if it has the chance to get back to an already-inflated balloon and inflate it further, while it is said to be *online* if it has to process the balloons sequentially, but is granted the knowledge of the balloon's capacity as soon as it is processed, regardless of whether or not it popped. Hence, an online blowing strategy must take irrevocable decisions in a scenario in which it has full knowledge of the set of capacities of the remaining balloons, while an offline blowing strategy has no particular restrictions on how to process the balloons, but it has to operate in a scenario with incomplete knowledge. Let  $\mathbf{OFF}_n$  and  $\mathbf{ON}_n$  be the *expected* total volume achievable by the optimal offline and online blowing strategy, respectively. For any fixed integer  $n \geq 1$ , denote as  $\mathcal{Y}_k$  the family of all subsets of  $\{1, 1/2, \dots, 1/n\}$  of size  $k$ , where each  $Y \in \mathcal{Y}_k$  is of the form  $\{1/y_1, \dots, 1/y_k\}$ , with  $y_1 < \dots < y_k$ ; so  $Y \in \mathcal{Y}_k$  is a set of  $k$  balloons listed in decreasing order.

Immorlica *et al.* [6] prove that under conditions *i)* and *ii)* stated above, for any set of  $n$  bidders with arbitrary utilities, the *expected revenue* of any mechanism is at most  $\mathbf{OFF}_n$  times the one raised by the best fixed-price scheme. In order to upper bound  $\mathbf{OFF}_n$ , they first show that  $\mathbf{OFF}_n \leq \mathbf{ON}_n$  for any positive integer  $n$ , then they determine the optimal online blowing strategy, thus proving that

$$\mathbf{ON}_n = \sum_{k=1}^n \sum_{Y \in \mathcal{Y}_k} \frac{\max_{1 \leq j \leq k} \{j/y_j\}}{\binom{n}{k} \cdot k}, \quad (1)$$

and finally they show that  $\mathbf{ON}_\infty \leq 4.331$ . Moreover, they prove that a *greedy mechanism*, that is, the offline blowing strategy which tries to blow up each balloon at the maximum possible capacity, achieves an expected volume equal to  $\sum_{i=1}^n 1/i^2$  which implies  $\mathbf{OFF}_\infty \geq \pi^2/6 \approx 1.6449$ .

Jung and Chwa [7] improved these bounds by designing an offline blowing strategy, called *Bunch*, yielding  $\mathbf{OFF}_\infty \geq 1.6595$  and proving that the right hand side of Equation (1) is at most  $2 - H_n/n$ , which yields  $\mathbf{ON}_\infty \leq 2$ . They also formulate a conjecture which, whenever true, would give  $\mathbf{ON}_n \geq 2 - (2H_n - 1)/n$ , thus implying  $\mathbf{ON}_\infty = 2$ .

**Our Contribution.** We further improve the lower bound on  $\mathbf{OFF}_\infty$  by designing an offline blowing strategy, that we call *Group<sub>k</sub>*. This strategy works by partitioning the  $n - 1$  smallest balloons into  $(n - 1)/k$  groups of size  $k$  and then

applying an ad-hoc (possibly optimal) strategy to the first of these groups and a simple basic blowing strategy to each of the remaining ones. The performance of  $Group_k$  improves when  $k$  increases and better lower bounds can be achieved by delving into a more detailed analysis covering higher values of  $k$ . For  $k = 5$ , we achieve  $\mathbf{OFF}_\infty \geq 1.68$ .

We also focus on the exact computation of  $\mathbf{ON}_n$  for any integer  $n \geq 1$ . To this aim, note that the characterization of  $\mathbf{ON}_n$  given by Immorlica *et al.* [6] through Equation (1) is neither easy to be analytically analyzed nor efficiently computable, since it requires to generate all subsets of a set of cardinality  $n$  whose number is exponential in  $n$ . We give an alternative exact formula for  $\mathbf{ON}_n$  which, although remaining of challenging analytical analysis, can be computed by an algorithm of running time  $O(n^5)$ . This allows us to disprove the conjecture formulated by Jung and Chwa [7] (which was experimentally verified only for  $n \leq 21$ ) as soon as  $n \geq 44$  and to provide an empirical evidence that  $\mathbf{ON}_\infty < 2$ . We hence **conjecture** that

$$\mathbf{ON}_n \leq \sum_{i=1}^n \frac{1}{i^2} + \frac{1}{2n} \sum_{i=2}^n \left( \frac{n-i}{n} - \frac{1}{i+1} \right).$$

The validity of this new conjecture, which thanks to our algorithm can be verified for  $n$  in the order of the hundreds, would imply  $\mathbf{ON}_\infty \leq \pi^2/6 + 1/4 \approx 1.8949$ . Such a value matches an experiment conducted by Immorlica *et al.* [6] who estimated  $\mathbf{ON}_{1000}$  by applying the optimal online blowing strategy on a sample of  $10^5$  random sequences of balloons.

**Paper Organization.** Next section contains the necessary definitions and notation. In Sections 3 and 4 we present the description and the analysis of the performance of the offline blowing strategy  $Group_k$  for  $k = 5$  and the algorithm for computing the exact value of  $\mathbf{ON}_n$ , respectively. Finally, in the last section we discuss conclusive remarks and open problems.

## 2 Definitions, Notation

For an integer  $k \geq 1$ , denote as  $[k]$  the set  $\{1, \dots, k\}$  and let  $Y_k = \{1/y_1, \dots, 1/y_k\}$ , with  $y_i \in [n]$  and  $y_i < y_{i+1}$  for any  $i \in [k-1]$ , be a *set of  $k$  balloons*. We denote as  $\mathcal{Y}_k$  the family of all sets of  $k$  balloons and with  $X_n = \{1, 1/2, \dots, 1/n\} \in \mathcal{Y}_n$  be the set of balloons defining the balloon popping problem.

Let  $\mathcal{B}(Y_k)$  be the set of all  $k!$  permutations of the  $k$  balloons in  $Y_k$ . For a permutation of balloons  $B \in \mathcal{B}(Y_k)$  and for any  $i \in [k]$ , we denote  $B_i$  as the  $i$ th balloon in  $B$  and  $B_{<i}$  as the set of the first  $i-1$  balloons in  $B$ , with  $B_{<1} := \emptyset$ . For a balloon  $j \in X_n$ , let  $pos_B(j)$  be the index  $i \in [n]$  such that  $B_i = j$ . Given a set of  $k$  balloons  $Y_k$ , we denote with  $first_B(Y_k)$  the balloon  $j' \in Y_k$  such that  $pos_B(j') = \min_{j \in Y_k} pos_B(j)$ , while, for any  $r \in [k]$ , we denote with  $suff_B^r(Y_k)$  the set of  $r$  balloons  $Y'_r \subseteq Y_k$  such that  $pos_B(first_B(Y'_r)) > pos_B(j)$

for any  $j \in Y_k \setminus Y'_r$ . For a balloon  $j \in X_n$  such that  $pos_B(j) = i > 1$  and a set of  $k$  balloons  $Y_k$ , we denote as  $pred_B(j, Y_k)$  the balloon  $j' \in Y_k$  such that  $pos_B(j') = \max_{l \in Y_k: pos_B(l) < i} pos_B(l)$ .

An online blowing strategy is a function  $s_{on} : \mathcal{Y}_k \mapsto \Re$ . Intuitively, an online blowing strategy specifies the capacity at which to blow up the current balloon, given that it belongs to the set of balloons  $Y_k$ . An execution of  $s_{on}$  on  $B$  is defined by the vector  $(c_1, \dots, c_n)$  such that  $c_i = s_{on}(X_n \setminus B_{<i})$ . Let  $I \subseteq [n]$  be the set of indexes  $i \in [n]$  for which  $c_i \leq B_i$ . The revenue of  $s_{on}$  on  $B$  is given by  $rev(s_{on}, B) = \sum_{i \in I} c_i$ , while the expected revenue of  $s_{on}$  on  $X_n$  is given by  $E[rev(s_{on}, B)] = \frac{1}{n!} \sum_{B \in \mathcal{B}(X_n)} rev(s_{on}, B)$ . We denote as  $s_{on}^*$  the optimal online blowing strategy and as  $\mathbf{ON}_n$  her expected revenue on  $X_n$ .

Let  $\mathcal{V}_n$  be the set of all  $n$ -dimensional vectors whose components belong to  $\Re \cup \{Popped\}$ . A vector  $V = (v_1, \dots, v_n) \in \mathcal{V}_n$  represents the current state of the  $n$  balloons, with  $v_i = Popped$  if the  $i$ th balloon has popped or  $v_i \in \Re$  if the  $i$ th balloon has been currently blown up to capacity  $v_i$ . An offline blowing strategy is a function  $s_{off} : \mathcal{B}(X_n) \mapsto \mathcal{V}_n$ . Intuitively, given a permutation of  $X_n$ , an offline blowing strategy outputs a final state in which some balloons are blown up to certain capacities and the remaining ones are popped. For a  $B \in \mathcal{B}(X_n)$  such that  $s_{off}(B) = V = (v_1, \dots, v_n) \in \mathcal{V}_n$ , let  $I \subseteq [n]$  be the set of indexes  $i \in [n]$  for which  $v_i \neq Popped$ . The revenue of  $s_{off}$  on  $B$  is given by  $rev(s_{off}, B) = \sum_{i \in I} v_i$ , while the expected revenue of  $s_{off}$  on  $X_n$  is given by  $E[rev(s_{off}, B)] = \frac{1}{n!} \sum_{B \in \mathcal{B}(X_n)} rev(s_{off}, B)$ . We denote  $s_{off}^*$  as the optimal online blowing strategy and  $\mathbf{OFF}_n$  as her expected revenue on  $X_n$ .

For a set of  $k$  balloons  $Y_k$ , denote as  $M(Y_k) = \max_{i \in [k]} \{i/y_i\}$ ,  $I(Y_k) = \{i \in [k] : i/y_i = M(Y_k)\}$ , and  $i^*(Y_k) = \min_{i \in I(Y_k)} \{i\}$ . Immorlica *et al.* [6] showed that  $s_{on}^*(Y_k) = \frac{1}{y_{i^*(Y_k)}}$ .

### 3 The Offline Blowing Strategy $Group_5$

For a pair of integers  $k \geq 2$  and  $i \in [|((n-1)/k)|]$ , let  $g_k(i) = \{j \in \mathbb{Z} : k(i-1) + 2 \leq j \leq ik + 1\}$  and  $group_k(i) = \{1/j \in X_n : j \in g_k(i)\}$ ; so, for  $k = 5$ ,  $group_5(1) = \{1/2, \dots, 1/6\}$ ,  $group_5(2) = \{1/7, \dots, 1/11\}$  and so on. Our offline blowing strategy  $Group_5(B)$  is defined as follows.

```

 $X := \{1, 1/2, \dots, 1/n\};$ 
for each  $i \in [n]$  do  $v_i := 0$ ;
for each  $i \in [n]$  do
|    $j := \max(X)$ ;
|   if  $j = 1$  then
|   |   blow the  $i$ th balloon of  $B$  up to capacity  $c := 1$ ;
|   else
|   |   let  $group_5(h)$  be the group including  $j$ ;
|   |    $r := |X \cap group_5(h)|$ ;
|   |   let  $a_1 > \dots > a_r$  be the ordered sequence of balloons in  $X \cap group_5(h)$ ;
|   |   if  $h = 1$  then  $c := strategy(r, a_1, \dots, a_r)$ ; else  $c := a_r$ ;

```

```

|   | blow the  $i$ th balloon of  $B$  up to capacity  $c$ ;
|   if the balloon pops at capacity  $c' < c$  then
|   |  $v_i := Popped$ ;
|   |  $X := X \setminus \{c'\}$ ;
|   else
|   |  $v_i := c$ ;
|   |  $X := X \setminus \{j\}$ ;

```

where the function  $strategy(r, a_1, \dots, a_r)$  is defined as follows.

```

if  $r = 2$  then
|   if  $a_1 \geq 3a_2/2$  then return  $a_1$ ; else return  $a_2$ ;
if  $r = 3$  then
|   if  $(a_1, a_2, a_3) = (1/4, 1/5, 1/6)$  then return  $1/6$ ;
|   else if  $a_1 + 3a_3 \geq 4a_2$  then return  $a_1$ ; else return  $a_2$ ;
if  $r = 4$  then
|   if  $a_1 - a_4 \geq 1/3$  then return  $a_1$ ; else return  $a_3$ ;
if  $r = 5$  then return  $a_3$ ;
return  $a_1$ ;

```

Note that, by definition,  $Group_5$  always blows the current balloon at capacity 1 as long as balloon 1 has not been processed yet. From that point on,  $Group_5$  will blow the current balloon to a capacity yielded by the group with the lowest index which has still at least one unprocessed balloon.

Let  $\mathbf{GR}_n$  denote the expected volume achieved by  $Group_5$  on a sequence of  $n$  balloons. By the definition of  $Group_5$ , it follows

$$\mathbf{GR}_n \geq 1 + \sum_{h=1}^{\lfloor (n-1)/5 \rfloor} \sum_{r=1}^5 \sum_{\{a_1, \dots, a_r\} \subseteq group_5(h)} (f(\{a_1, \dots, a_r\}) Pr[\{a_1, \dots, a_r\}]),$$

where  $f(\{a_1, \dots, a_r\})$  denotes the expected volume achieved by  $Group_5$  when processing the set of  $r$  balloons  $\{a_1, \dots, a_r\}$  and  $Pr[\{a_1, \dots, a_r\}]$  is the probability that such a set is processed by  $Group_5$ .

For the case of  $h = 1$ , that is for any set of  $r$  balloons  $\{a_1, \dots, a_r\} \subseteq \{1/2, \dots, 1/6\}$ , with  $r \in [5]$ , the value  $f(\{a_1, \dots, a_r\})$  is provided by the following Lemma.

**Lemma 1.** *The values for  $f(\{a_1, \dots, a_r\})$  reported in Table 1 hold.*

*Proof.* Function  $strategy(1, a_1)$  returns the value  $a_1$ , hence the claim holds for  $f(\{a_1\})$  when  $\{a_1\} \in \{\{1/2\}, \dots, \{1/6\}\}$ .

Function  $strategy(2, a_1, a_2)$  returns the value  $a_2$  when  $\{a_1, a_2\} \in \{\{1/3, 1/4\}, \{1/4, 1/5\}, \{1/5, 1/6\}\}$  and the value  $a_1$  in the remaining cases. Since it holds  $Pr[first_B(\{a_1, a_2\}) \geq a_1] = 1/2$  and  $Pr[first_B(\{a_1, a_2\}) \geq a_2] = 1$ , it follows that  $f(\{a_1, a_2\}) = a_2 + f(\{a_2\}) = 2a_2$  when  $\{a_1, a_2\} \in \{\{1/3, 1/4\}, \{1/4, 1/5\}, \{1/5, 1/6\}\}$  and  $f(\{a_1, a_2\}) = \frac{1}{2}(a_1 + f(\{a_2\})) + \frac{1}{2}f(\{a_1\}) = a_1 + \frac{a_2}{2}$  in the remaining cases.

Function  $strategy(3, a_1, a_2, a_3)$  returns the value  $a_1$  when  $\{a_1, a_2, a_3\} \in \{\{1/2, 1/4, 1/5\}, \{1/2, 1/4, 1/6\}, \{1/2, 1/5, 1/6\}, \{1/3, 1/5, 1/6\}\}$ , the value  $a_3$  when  $\{a_1, a_2, a_3\} = \{1/4, 1/5, 1/6\}$  and the value  $a_2$  in the remaining cases. Since it holds  $Pr[first_B(\{a_1, a_2, a_3\}) \geq a_1] = 1/3$ ,  $Pr[first_B(\{a_1, a_2, a_3\}) \geq a_2] = 2/3$  and  $Pr[first_B(\{a_1, a_2, a_3\}) \geq a_3] = 1$ , it follows that  $f(\{a_1, a_2, a_3\}) = \frac{a_1}{3} + \frac{1}{3}(f(\{a_1, a_2\}) + f(\{a_1, a_3\}) + f(\{a_2, a_3\}))$  when  $\{a_1, a_2, a_3\} \in \{\{1/2, 1/4, 1/5\}, \{1/2, 1/4, 1/6\}, \{1/2, 1/5, 1/6\}, \{1/3, 1/5, 1/6\}\}$ ,  $f(\{1/4, 1/5, 1/6\}) = \frac{1}{6} + f(\{1/5, 1/6\}) = \frac{1}{2}$  and  $f(\{a_1, a_2, a_3\}) = \frac{2a_2}{3} + \frac{2}{3}f(\{a_2, a_3\}) + \frac{1}{3}f(\{a_1, a_2\})$  in the remaining cases.

Function  $strategy(4, a_1, a_2, a_3, a_4)$  returns the value  $a_1$  when  $\{a_1, a_2, a_3, a_4\} \in \{\{1/2, 1/3, 1/4, 1/6\}, \{1/2, 1/3, 1/5, 1/6\}, \{1/2, 1/4, 1/5, 1/6\}\}$  and the value  $a_3$  in the remaining cases. Since it holds  $Pr[first_B(\{a_1, a_2, a_3, a_4\}) \geq a_1] = 1/4$  and  $Pr[first_B(\{a_1, a_2, a_3, a_4\}) \geq a_3] = 3/4$ , it follows that  $f(\{a_1, a_2, a_3\}) = \frac{a_1}{4} + \frac{1}{4}(f(\{a_1, a_2, a_3\}) + f(\{a_1, a_2, a_4\}) + f(\{a_1, a_3, a_4\}) + f(\{a_2, a_3, a_4\}))$  when  $\{a_1, a_2, a_3, a_4\} \in \{\{1/2, 1/3, 1/4, 1/6\}, \{1/2, 1/3, 1/5, 1/6\}, \{1/2, 1/4, 1/5, 1/6\}\}$  and  $f(\{a_1, a_2, a_3, a_4\}) = \frac{3a_3}{4} + \frac{3}{4}f(\{a_2, a_3, a_4\}) + \frac{1}{4}f(\{a_1, a_2, a_3\})$  in the remaining cases.

Function  $strategy(5, 1/2, 1/3, 1/4, 1/5, 1/6)$  returns the value  $1/4$ . Hence, since it holds  $Pr[first_B(\{1/2, 1/3, 1/4, 1/5, 1/6\}) \geq 1/4] = 3/5$ , it follows that  $f(\{1/2, 1/3, 1/4, 1/5, 1/6\}) = \frac{3}{20} + \frac{3}{5}f(\{1/3, 1/4, 1/5, 1/6\}) + \frac{1}{5}f(\{1/2, 1/3, 1/4, 1/5\}) + \frac{1}{5}f(\{1/2, 1/3, 1/4, 1/6\}) = \frac{6359}{7200}$ .  $\square$

$\{a_1, \dots, a_r\}$	$f(\{a_1, \dots, a_r\})$	$\{a_1, \dots, a_r\}$	$f(\{a_1, \dots, a_r\})$	$\{a_1, \dots, a_r\}$	$f(\{a_1, \dots, a_r\})$
$\frac{1}{i}$	$\frac{1}{i}$	$(\frac{1}{4}, \frac{1}{6})$	$\frac{1}{3}$	$(\frac{1}{3}, \frac{1}{4}, \frac{1}{6})$	$\frac{5}{9}$
$(\frac{1}{2}, \frac{1}{3})$	$\frac{2}{3}$	$(\frac{1}{5}, \frac{1}{6})$	$\frac{1}{3}$	$(\frac{1}{3}, \frac{1}{5}, \frac{1}{6})$	$\frac{91}{180}$
$(\frac{1}{2}, \frac{1}{4})$	$\frac{5}{8}$	$(\frac{1}{2}, \frac{1}{3}, \frac{1}{4})$	$\frac{7}{9}$	$(\frac{1}{4}, \frac{1}{5}, \frac{1}{6})$	$\frac{1}{2}$
$(\frac{1}{2}, \frac{1}{5})$	$\frac{3}{5}$	$(\frac{1}{2}, \frac{1}{3}, \frac{1}{5})$	$\frac{11}{15}$	$(\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5})$	$\frac{599}{720}$
$(\frac{1}{2}, \frac{1}{6})$	$\frac{7}{12}$	$(\frac{1}{2}, \frac{1}{3}, \frac{1}{6})$	$\frac{13}{18}$	$(\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{6})$	$\frac{233}{288}$
$(\frac{1}{3}, \frac{1}{4})$	$\frac{1}{2}$	$(\frac{1}{2}, \frac{1}{4}, \frac{1}{5})$	$\frac{17}{24}$	$(\frac{1}{2}, \frac{1}{3}, \frac{1}{5}, \frac{1}{6})$	$\frac{47}{60}$
$(\frac{1}{3}, \frac{1}{5})$	$\frac{13}{30}$	$(\frac{1}{2}, \frac{1}{4}, \frac{1}{6})$	$\frac{49}{72}$	$(\frac{1}{2}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6})$	$\frac{551}{720}$
$(\frac{1}{3}, \frac{1}{6})$	$\frac{5}{12}$	$(\frac{1}{2}, \frac{1}{5}, \frac{1}{6})$	$\frac{121}{180}$	$(\frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6})$	$\frac{27}{40}$
$(\frac{1}{4}, \frac{1}{5})$	$\frac{2}{5}$	$(\frac{1}{3}, \frac{1}{4}, \frac{1}{5})$	$\frac{3}{5}$	$(\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6})$	$\frac{6359}{7200}$

**Fig. 1.** The values  $f(\{a_1, \dots, a_r\})$  for any set of  $r$  balloons  $\{a_1, \dots, a_r\} \in group_5(1)$ , with  $r \in [5]$

Given a set  $U$  of  $n$  elements, let  $S(U)$  be a random permutation of the  $n$  elements of  $U$ , where each permutation is equally likely. The following result will be of crucial importance in the computation of the quantities  $Pr[\{a_1, \dots, a_r\}]$  for any set of  $r$  balloons  $\{a_1, \dots, a_r\} \in group_5(h)$ , with  $r \in [5]$ .

**Lemma 2.** For any triple of pairwise disjoint sets  $I, J, K \subseteq U$ , with  $I, K \neq \{\emptyset\}$ , the probability that in  $S(U)$  each element of  $K$  follows all the elements of  $I \cup J$  and that at least one element of  $I$  follows all the elements of  $J$  is  $\frac{|I|(|I|+|J|-1)!|K|!}{(|I|+|J|+|K|)!}$ .

We can now compute the contribution of  $group_5(1)$  to  $\mathbf{GR}_n$ .

**Lemma 3.** For any  $n \geq 6$ , the contribution of the balloons belonging to  $group_5(1)$  to  $\mathbf{GR}_n$  is  $\frac{3679}{7200}$ .

*Proof.* Set  $Y = \{1, 1/2, \dots, 1/6\}$ .

The case of  $r = 1$  and  $\{a\} \subset group_5(1)$ , occurs when  $suff_B^1(Y) = \{a\}$  and  $pred_B(a, Y) = 1$ . Because of Lemma 2, the probability that this event happens is  $1/30$ . Hence, the expected contribution of the case  $r = 1$  to  $Group_5$  for the balloons belonging to  $group_5(1)$  is  $\frac{1}{30} \sum_{\{a\} \subset group_5(1)} f(\{a\}) = \frac{1}{30} (\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6})$  because of Lemma 1.

The case of  $r = 2$  and  $\{a_1, a_2\} \subset group_5(1)$  occurs when  $suff_B^2(Y) = \{a_1, a_2\}$  and  $pred_B(first_B(\{a_1, a_2\}), Y) = 1$ . Because of Lemma 2, the probability that this event happens is  $1/60$ . Hence, the expected contribution of the case  $r = 2$  to  $Group_5$  for the balloons belonging to  $group_5(1)$  is  $\frac{1}{60} \sum_{\{a_1, a_2\} \subset group_5(1)} f(\{a_1, a_2\}) = \frac{1}{60} (\frac{2}{3} + \frac{5}{8} + \frac{3}{5} + \frac{7}{12} + \frac{1}{2} + \frac{13}{30} + \frac{5}{12} + \frac{2}{5} + \frac{1}{3} + \frac{1}{3})$  because of Lemma 1.

The case of  $r = 3$  and  $\{a_1, a_2, a_3\} \subset group_5(1)$  occurs when  $suff_B^3(Y) = \{a_1, a_2, a_3\}$  and  $pred_B(first_B(\{a_1, a_2, a_3\}), Y) = 1$ . Because of Lemma 2, the probability that this event happens is  $1/60$ . Hence, the expected contribution of the case  $r = 3$  to  $Group_5$  for the balloons belonging to  $group_5(1)$  is  $\frac{1}{60} \sum_{\{a_1, a_2, a_3\} \subset group_5(1)} f(\{a_1, a_2, a_3\}) = \frac{1}{60} (\frac{7}{9} + \frac{11}{15} + \frac{13}{18} + \frac{17}{24} + \frac{49}{72} + \frac{121}{180} + \frac{3}{5} + \frac{5}{9} + \frac{91}{180} + \frac{1}{2})$  because of Lemma 1.

The case of  $r = 4$  and  $\{a_1, a_2, a_3, a_4\} \subset group_5(1)$  occurs when  $suff_B^4(Y) = \{a_1, a_2, a_3, a_4\}$  and  $pred_B(first_B(\{a_1, a_2, a_3, a_4\}), Y) = 1$ . Because of Lemma 2, the probability that this event happens is  $1/30$ . Hence, the expected contribution of the case  $r = 4$  to  $Group_5$  for the balloons belonging to  $group_5(1)$  is  $\frac{1}{30} \sum_{\{a_1, a_2, a_3, a_4\} \subset group_5(1)} f(\{a_1, a_2, a_3, a_4\}) = \frac{1}{30} (\frac{599}{720} + \frac{233}{288} + \frac{47}{60} + \frac{551}{720} + \frac{27}{40})$  because of Lemma 1.

The case of  $r = 5$  occurs when  $first_B(Y) = \{1\}$ . Because of Lemma 2, the probability that this event happens is  $1/6$ . Hence, the expected contribution of the case  $r = 5$  to  $Group_5$  for the balloons belonging to  $group_5(1)$  is  $\frac{1}{6} f(\{1/2, \dots, 1/6\}) = \frac{1}{6} \cdot \frac{6359}{7200}$  because of Lemma 1.

By summing up all the contributions for  $r \in [5]$ , the claim follows.  $\square$

We can now proceed with the evaluation of the performance of  $Group_5$  for  $n$  going to infinity.

**Theorem 1.**  $\mathbf{GR}_\infty > 1.68$ .

*Proof.* By exploiting Lemma 3, we have that

$$\mathbf{GR}_n \geq \frac{10897}{7200} + \sum_{h=2}^{\lfloor (n-1)/5 \rfloor} \sum_{r=1}^5 \sum_{\{a_1, \dots, a_r\} \in group_5(h)} (f(\{a_1, \dots, a_r\}) Pr[\{a_1, \dots, a_r\}]).$$

Consider now  $group_5(h) = \left\{ \frac{1}{5h-3}, \frac{1}{5h-2}, \frac{1}{5h-1}, \frac{1}{5h}, \frac{1}{5h+1} \right\}$  with  $h \geq 2$  and set  $Y = \{1, 1/2, \dots, 1/(5h+1)\}$ .

The case of  $r = 1$  and  $\{a\} \subset group_5(h)$  occurs when  $suff_B^1(Y) = \{a\}$  and  $pred_B(a, Y) \notin group_5(h)$ . Because of Lemma 2, the probability that this event happens is  $\frac{5h-4}{5h(5h+1)}$ . Clearly, in such a case, it holds  $f(\{a\}) = a$ . Hence, the expected contribution of the case  $r = 1$  to  $Group_5$  for the balloons belonging to  $group_5(h)$  for  $h \geq 2$  is  $\frac{5h-4}{5h(5h+1)} \left( \frac{1}{5h-3} + \frac{1}{5h-2} + \frac{1}{5h-1} + \frac{1}{5h} + \frac{1}{5h+1} \right)$ .

The case of  $r = 2$  and  $\{a_1, a_2\} \subset group_5(h)$  occurs when  $suff_B^2(Y) = \{a_1, a_2\}$  and  $pred_B(first_B(\{a_1, a_2\}), Y) \notin group_5(h)$ . Because of Lemma 2, the probability that this event happens is  $\frac{2(5h-4)}{(5h-1)5h(5h+1)}$ . By definition of  $Group_5$ , in such a case, it holds  $f(\{a_1, a_2\}) = 2a_2$ . By considering all possible 10 sets  $\{a_1, a_2\} \subset group_5(h)$ , it follows that the expected contribution of the case  $r = 2$  to  $Group_5$  for the balloons belonging to  $group_5(h)$  for  $h \geq 2$  is  $\frac{2(5h-4)}{(5h-1)5h(5h+1)} \left( \frac{2}{5h-2} + \frac{4}{5h-1} + \frac{6}{5h} + \frac{8}{5h+1} \right)$ .

The case of  $r = 3$  and  $\{a_1, a_2, a_3\} \subset group_5(h)$  occurs when  $suff_B^3(Y) = \{a_1, a_2, a_3\}$  and  $pred_B(first_B(\{a_1, a_2, a_3\}), Y) \notin group_5(h)$ . Because of Lemma 2, the probability that this event happens is  $\frac{6(5h-4)}{(5h-2)(5h-1)5h(5h+1)}$ . By definition of  $Group_5$ , in such a case, it holds  $f(\{a_1, a_2, a_3\}) = 3a_3$ . By considering all possible 10 sets  $\{a_1, a_2, a_3\} \subset group_5(h)$ , it follows that the expected contribution of the case  $r = 3$  to  $Group_5$  for the balloons belonging to  $group_5(h)$  for  $h \geq 2$  is  $\frac{6(5h-4)}{(5h-2)(5h-1)5h(5h+1)} \left( \frac{3}{5h-1} + \frac{9}{5h} + \frac{18}{5h+1} \right)$ .

The case of  $r = 4$  and  $\{a_1, a_2, a_3, a_4\} \subset group_5(h)$  occurs when  $suff_B^4(Y) = \{a_1, a_2, a_3, a_4\}$  and  $pred_B(first_B(\{a_1, a_2, a_3, a_4\}), Y) \notin group_5(h)$ . Because of Lemma 2, the probability that this event happens is  $\frac{24(5h-4)}{(5h-3)(5h-2)(5h-1)5h(5h+1)}$ . By definition of  $Group_5$ , in such a case, it holds  $f(\{a_1, a_2, a_3, a_4\}) = 4a_4$ . By considering all possible 5 sets  $\{a_1, a_2, a_3, a_4\} \subset group_5(h)$ , it follows that the expected contribution of the case  $r = 4$  to  $Group_5$  for the balloons belonging to  $group_5(h)$  for  $h \geq 2$  is  $\frac{24(5h-4)}{(5h-3)(5h-2)(5h-1)5h(5h+1)} \left( \frac{4}{5h} + \frac{16}{5h+1} \right)$ .

The case of  $r = 5$  occurs when  $suff_B^5(Y) = group_5(h)$ . Because of Lemma 2, the probability that this event happens is  $\frac{120}{(5h-3)(5h-2)(5h-1)5h(5h+1)}$ . By definition of  $Group_5$ , in such a case, it holds  $f(group_5(h)) = 5a_5$ . Hence, it follows that the expected contribution of the case  $r = 5$  to  $Group_5$  for the balloons belonging to  $group_5(h)$  for  $h \geq 2$  is  $\frac{120}{(5h-3)(5h-2)(5h-1)5h(5h+1)} \cdot \frac{5}{5h+1}$ .

By summing up all the contribution for  $r = [5]$ , we obtain

$$\mathbf{GR}_n \geq \frac{10897}{7200} + \sum_{h=2}^{\lfloor (n-1)/5 \rfloor} \frac{625h^4 - 250h^3 - 175h^2 + 60h + 8}{h(5h-3)(5h-2)(5h-1)^2(5h+1)}.$$

Note that the quantity  $\frac{625h^4 - 250h^3 - 175h^2 + 60h + 8}{h(5h-3)(5h-2)(5h-1)^2(5h+1)}$  is non-negative for any  $h \geq 0$ . Hence, it holds

$$\mathbf{GR}_\infty \geq \mathbf{GR}_{5001} = \frac{10897}{7200} + \sum_{h=2}^{1000} \frac{625h^4 - 250h^3 - 175h^2 + 60h + 8}{h(5h-3)(5h-2)(5h-1)^2(5h+1)} > 1.68.$$

□

## 4 An Algorithm for Computing $\mathbf{ON}_n$

For any  $k, j \in [n]$  and  $h \in [k]$ , define  $\mathcal{T}_k(j, h) = \{Y_k \in \mathcal{Y}_k : i^*(Y_k) = h \text{ and } y_h = 1/j\}$  and denote as  $t_k(j, h) = |\mathcal{T}_k(j, h)|$ . We give the following new characterization of  $\mathbf{ON}_n$ .

$$\mathbf{Theorem 2.} \quad \mathbf{ON}_n = \sum_{j=1}^n \sum_{k=1}^n \left( \frac{1}{\binom{n}{k} \cdot k \cdot j} \sum_{h=\max\{1, k+j-n\}}^{\min\{j, k\}} (h \cdot t_k(j, h)) \right).$$

*Proof.* For a fixed  $Y_k \in \mathcal{Y}_k$ , it holds  $Y_k \in \mathcal{T}_k(j, h)$  if and only if *i*)  $\frac{p}{y_p} < \frac{h}{j}$  for any  $p \in [h-1]$ , and *ii*)  $\frac{p}{y_p} \leq \frac{h}{j}$  for any  $h+1 \leq p \leq k$ . Note that, by the definitions of  $h$ , there must be exactly  $h-1$  balloons before balloon  $1/j$  and exactly  $k-h$  balloons after balloon  $1/j$  in  $Y_k$ . By the definition of  $Y_k$ , it follows  $h \leq j$  and  $h \geq k+j-n$ . These two constraints, together with  $h \in [k]$ , implies  $\max\{1, k+j-n\} \leq h \leq \min\{k, j\}$ .

Let  $\mathcal{T}_{big}(k, j, h)$  be the family of all sets of  $h-1$  balloons satisfying condition *i*) and  $\mathcal{T}_{small}(k, j, h)$  be the family of all sets of  $k-h$  balloons satisfying condition *ii*). It follows that  $t_k(j, h) = |\mathcal{T}_{big}(k, j, h)| \cdot |\mathcal{T}_{small}(k, j, h)|$  (note that, when  $h-1=0$ , it holds  $|\mathcal{T}_{big}(k, j, h)|=1$  since  $\mathcal{T}_{big}(k, j, h)=\emptyset$  and similarly, when  $k-h=0$ , it holds  $|\mathcal{T}_{small}(k, j, h)|=1$  since  $\mathcal{T}_{small}(k, j, h)=\emptyset$ ).

For any set of  $k$  balloons  $Y_k \in \mathcal{T}_k(j, h)$ , the probability that the optimal online blowing strategy  $s^*(Y_k) = 1/j$  succeeds in a randomly permuted sequence of the  $k$  balloons in  $Y_k$  is  $h/k$ , that is, equal to the probability that the first balloon of the sequence is one of the  $h$  balloons in  $Y_k$  with capacity at least  $1/j$ . Thus, the expected contribution of the optimal online blowing strategy  $1/j$  when restricted to sequences of  $k$  balloons is equal to

$$\sum_{h=\max\{1, k+j-n\}}^{\min\{j, k\}} \frac{h \cdot t_k(j, h)}{k \cdot j}.$$

It follows that the expected contribution of the optimal online blowing strategy  $1/j$  on all possible sequences of balloons is equal to

$$\sum_{k=1}^n \left( \frac{1}{\binom{n}{k}} \sum_{h=\max\{1, k+j-n\}}^{\min\{j, k\}} \frac{h \cdot t_k(j, h)}{k \cdot j} \right).$$

By summing up the expected contributions for any  $j \in [n]$ , the claim follows. □

We now give a polynomial time algorithm for computing the quantity  $t_k(j, h)$ . Note that a set of  $h - 1$  balloons  $Y_{h-1} \in \mathcal{T}_{big}(k, j, h)$  if and only if  $y_i \geq \lfloor \frac{i \cdot j}{h} \rfloor + 1$ . Thus,  $|\mathcal{T}_{big}(k, j, h)|$  coincides with the number of  $(h - 1)$ -tuples  $(t_1, \dots, t_{h-1})$  such that  $\lfloor \frac{i \cdot j}{h} \rfloor + 1 \leq t_i \leq j - h + i$  for any  $i \in [h - 1]$  and  $t_i < t_{i+1}$  for any  $i \in [h - 2]$ . Similarly, a set of  $k - h$  balloons  $Y_{k-h} \in \mathcal{T}_{small}(k, j, h)$  if and only if  $y_i \geq \lceil \frac{(h+i) \cdot j}{h} \rceil$ . Thus,  $|\mathcal{T}_{small}(k, j, h)|$  coincides with the number of  $(k - h)$ -tuples  $(t_1, \dots, t_{k-h})$  such that  $\lceil \frac{(h+i) \cdot j}{h} \rceil \leq t_i \leq n - k + h + i$  for any  $i \in [k - h]$  and  $t_i < t_{i+1}$  for any  $i \in [k - h - 1]$ . Hence, in order to compute  $t_k(j, h)$ , we need an algorithm for solving the following problem:

Given  $r$  positive integers  $a_1, \dots, a_r$  such that  $a_i < a_{i+1}$  for any  $i \in [r - 1]$  and  $a_i \leq b - r + i$  for some  $b \in \mathbb{Z}_{>0}$ , let  $\mathcal{T}$  be the set of  $r$ -tuples  $(t_1, \dots, t_r)$  such that  $a_i \leq t_i \leq b - r + i$  for any  $i \in [r]$  and  $t_i < t_{i+1}$  for any  $i \in [r - 1]$ . Which is the value of  $|\mathcal{T}|$ ?

The solution to this problem can be constructed as follows. For any  $p \in [r]$ , let  $\mathcal{T}_p$  be the set of  $p$ -tuples  $(t'_{r-p+1}, \dots, t'_r)$  such that there exists a tuple  $(t_1, \dots, t_{r-p}, t'_{r-p+1}, \dots, t'_r) \in \mathcal{T}$ . For any  $i \in \{a_1, a_1 + 1, \dots, b\}$ , let  $t_p(i)$  be the number of  $p$ -tuples  $(t'_{r-p+1}, \dots, t'_r) \in \mathcal{T}_p$  such that  $t'_{r-p+1} = i$ . It holds  $|\mathcal{T}| = \sum_{i \in \{a_1, \dots, b-r+1\}} t_r(i)$ , where

$$t_p(i) = \begin{cases} 1 & \text{if } p = 1 \wedge i \in \{a_r, \dots, b\}, \\ \sum_{j > i} t_{p-1}(j) & \text{if } 1 < p \leq r \wedge i \in \{a_{r-p+1}, \dots, b - p + 1\}, \\ 0 & \text{otherwise.} \end{cases}$$

The computation of  $t_p(i)$  by a naïve algorithm requires a running time of  $O(n^3)$ . Anyway, it is possible to adopt a dynamic programming algorithm by using the following alternative characterization for  $t_p(i)$ .

$$t_p(i) = \begin{cases} 1 & \text{if } p = 1 \wedge i \in \{a_r, \dots, b\}, \\ t_p(i+1) + t_{p-1}(i+1) & \text{if } 1 < p \leq r \wedge i \in \{a_{r-p+1}, \dots, b - p + 1\}, \\ 0 & \text{otherwise.} \end{cases}$$

With this approach, the computation of  $t_p(i)$  and hence of the quantities  $t_k(j, h)$  can be done in time  $O(n^2)$  for any triple of integers  $j, k, h$ . Recalling that  $j, k \in [n]$  and  $\max\{1, k + j - n\} \leq h \leq \min\{j, k\}$ , the overall computation of  $\mathbf{ON}_n$  can be performed in time  $O(n^5)$  in the worst-case.

The quantities  $\mathbf{ON}_n$  for some values of  $n$  are reported in Figure 2.

Jung and Chwa [7] conjectured that  $\mathbf{ON}_n \geq \mathbf{OLD-CON}_n := 2 - \frac{2H_n - 1}{n}$  based on a property that they could experimentally verify up to  $n = 21$ . As it can be appreciated from the values reported in Figure 2, their conjecture, which would have implied,  $\mathbf{ON}_\infty = 2$ , is not valid and it is disproved as soon as  $n \geq 44$ .

The growth rate of  $\mathbf{ON}_n$  witnessed by our algorithm strongly encourages the belief that  $\mathbf{ON}_\infty < 2$ . To this aim, we formulate here the following new conjecture that, as shown in Figure 2, is verified for  $n \leq 600$ :

$$\mathbf{ON}_n \leq \mathbf{NEW-CON}_n := \sum_{i=1}^n \frac{1}{i^2} + \frac{1}{2n} \sum_{i=2}^n \left( \frac{n-i}{n} - \frac{1}{i+1} \right).$$

If our conjecture holds then, by taking the limit for  $n$  going to infinity of  $\mathbf{NEW-CON}_n$ , it follows  $\mathbf{ON}_\infty \leq \pi^2/6 + 1/4 \approx 1.8949$ . It is interesting to note that this value matches an experiment conducted by Immorlica *et al.* [6] who estimated  $\mathbf{ON}_{1000}$  by applying the optimal online blowing strategy on a sample of  $10^5$  random sequences of balloons.

$n$	$\mathbf{ON}_n$	$\mathbf{OLD-CON}_n$	$\mathbf{NEW-CON}_n$	$n$	$\mathbf{ON}_n$	$\mathbf{OLD-CON}_n$	$\mathbf{NEW-CON}_n$
1	1	1	1	20	1.763399	1.690226	1.76872
2	1.25	1	1.25	43	1.822355	1.82093	1.827179
3	1.388889	1.111111	1.388889	44	1.823642	1.823967	1.828455
4	1.472222	1.208333	1.475694	50	1.830404	1.840032	1.835141
5	1.532222	1.286667	1.535278	100	1.857147	1.906252	1.861547
6	1.573889	1.35	1.578889	200	1.872309	1.94622	1.876501
7	1.607766	1.402041	1.612307	300	1.877853	1.961449	1.881969
8	1.633639	1.445536	1.638806	400	1.880774	1.96965	1.88485
9	1.655153	1.482451	1.660381	500	1.882592	1.974829	1.886643
10	1.672889	1.514206	1.678319	600	1.883838	1.978417	1.887873

**Fig. 2.** The values of  $\mathbf{ON}_n$ ,  $\mathbf{OLD-CON}_n$  and  $\mathbf{NEW-CON}_n$  for some meaningful values of  $n$

## 5 Conclusions and Open Problems

We revisited the balloon popping problem introduced by Immorlica *et al.* [6] and later reconsidered by Jung and Chwa [7]. We improved the lower bound on  $\mathbf{OFF}_\infty$  from 1.6595 to 1.68 by designing and analyzing the offline blowing strategy *Group*<sub>5</sub>. The ingredients needed for the analysis of *Group*<sub>5</sub> suffice to bound the performance of the offline blowing strategy *Group*<sub>k</sub> for any  $k \geq 2$ . Hence, improvements on our 1.68 lower bound can be achieved at the expenses of the wider case analysis yielded by higher values of  $k$ .

While the determination of the optimal offline blowing strategy remains a challenging open question, the online blowing strategy was exactly characterized by Immorlica *et al.* [6] and from such a characterization, they derived an exact, yet complex, formula for the value  $\mathbf{ON}_n$ . Using this formula, Jung and Chwa [7] proved that  $\mathbf{ON}_\infty \leq 2$ . They further proposed a conjecture which would imply  $\mathbf{ON}_\infty \geq 2$  and left its proof as an open question. We disproved such a conjecture by providing a new exact formula for  $\mathbf{ON}_n$  which, differently from the one proposed by Immorlica *et al.* [6], can be evaluated by means of an algorithm

of running time  $O(n^5)$ . In fact, our algorithm shows that the conjecture of Jung and Chwa [7] cannot hold for any  $n \geq 44$ , but, more importantly, it reveals a growth ratio for  $\mathbf{ON}_n$  which strongly encourages the hypothesis that  $\mathbf{ON}_n$  might converge to some value strictly smaller than 2. We then propose a new conjecture, which upper bounds  $\mathbf{ON}_n$  for any  $n \geq 1$  and whose validity could be tested through our algorithm up to  $n = 600$ , according to which it would follow  $\mathbf{ON}_\infty \leq \pi^2/6 + 1/4 \approx 1.8949$ . The proof of our conjecture is a major open problem and would provide a significant improvement on the upper bound for the balloon popping problem.

## References

1. Balcan, M.F., Blum, A., Mansour, Y.: Item pricing for revenue maximization. In: Proceedings of the 9th ACM Conference on Electronic Commerce (EC), pp. 50–59. ACM Press (2008)
2. Bar-Yossef, Z., Hildrum, K., Wu, F.: Incentive-compatible online auctions for digital goods. In: Proceedings of 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 964–970. ACM/SIAM Press (2002)
3. Blum, A., Hartline, J.: Near-optimal online auctions. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1156–1163. ACM/SIAM Press (2005)
4. Borgs, C., Chayes, J., Immorlica, N., Mahdian, M., Saberi, A.: Multi-unit auctions with budget constrained bidders. In: Proceedings of 6th ACM Conference on Electronic Commerce (EC), pp. 44–51. ACM Press (2005)
5. Goldberg, A., Hartline, J., Wright, A.: Competitive auctions and digital goods. In: Proceedings of 12th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 735–744. ACM/SIAM Press (2001)
6. Immorlica, N., Karlin, A.R., Mahdian, M., Talwar, K.: Balloon Popping With Applications to Ascending Auctions. In: Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 104–112. IEEE Computer Society (2007)
7. Jung, H., Chwa, K.-Y.: The Balloon Popping Problem Revisited: Lower and Upper Bounds. Theory of Computing Systems 49(1), 182–195 (2011)

# On the Sequential Price of Anarchy of Isolation Games

Anna Angelucci<sup>1</sup>, Vittorio Bilò<sup>2</sup>, Michele Flammini<sup>1</sup>, and Luca Moscardelli<sup>3</sup>

<sup>1</sup> Department of Information Engineering, Computer Science and Mathematics,  
University of L’Aquila

Via Vetoio, Loc. Coppito, 67100 L’Aquila, Italy  
`{anna.angelucci,michele.flammini}@univaq.it`

<sup>2</sup> Department of Mathematics and Physics “Ennio De Giorgi”, University of Salento  
Provinciale Lecce-Arnesano P.O. Box 193, 73100 Lecce, Italy  
`vittorio.bilo@unisalento.it`

<sup>3</sup> Department of Economic Studies, University of Chieti-Pescara  
Viale Pindaro 42, 65127 Pescara, Italy  
`luca.moscardelli@unich.it`

**Abstract.** We study the performance of Subgame Perfect Equilibria, a solution concept which better captures the players’ rationality in sequential games with respect to the classical myopic dynamics based on the notions of improving deviations and Nash Equilibria, in the context of sequential isolation games. In particular, for two important classes of sequential isolation games, we show upper and lower bounds on the Sequential Price of Anarchy, that is the worst-case ratio between the social performance of an optimal solution and that of a Subgame Perfect Equilibrium, under the two classical social functions mostly investigated in the scientific literature, namely, the minimum utility per player and the sum of the players’ utilities.

## 1 Introduction

In competitive location games [8] players aim at choosing suitable locations or points in given metric spaces so as to maximize their utility or revenue. Depending on different parameters such as the underlying metric space, the number of players, the adopted solution concept, the customers’ behavior and so on, several scenarios arise.

In this paper we consider *isolation games* [19], a class of competitive location games in which the utility of a player is defined as a function of her distances from the other ones in an underlying edge-weighted graph. For example, it is natural to define the utility of a player as the distance from the nearest one (*nearest-neighbor isolation game*), or as the sum of the distances from all the other players (*total-distance isolation game*).

Isolation games find a natural application in data clustering [10] and geometric sampling [18]. Moreover, as pointed out in [19], they can be used to obtain a good approximation of the strategy a player should compute in another competitive

location game, called Voronoi game [6,7,14], which is among the most studied competitive location games. Here, the utility of a player is given by the total number of all points that are closer to her than to any other player (Voronoi area), where points which are equidistant to several players are evenly split up among them. As another interesting field of application for isolation games, consider the following problem in non-cooperative wireless network design: We are given a set of users who have to select a spot where to locate an antenna so as to transmit and receive a radio signal. When more than just one antenna is transmitting contemporaneously, interference between two or more signals may occur. If users are selfish players interested in minimizing the amount of interference their antenna will be subject to, they will decide to locate it as far as possible from the other ones thus giving rise to a particular isolation game.

The fundamental approach adopted in the literature in order to model the non-cooperative behavior of selfish players has been that of resorting on different concepts of equilibrium to characterize stable solutions, Nash Equilibrium being among the most investigated ones [15]. In such a setting, the problem of measuring the loss of optimality due to the selfishness of the players becomes of crucial importance. To this aim, given a social function measuring the overall efficiency of any solution of the game, the notions of Price of Anarchy (PoA) [11], obtained by comparing the social value of the worst Nash Equilibrium with that of the social optimum, and Price of Stability (PoS) [1], obtained by comparing the social value of the best Nash Equilibrium with that of the social optimum, have been widely used in the Algorithmic Game Theory literature, see [16]. Nevertheless, these two metrics naturally extend to other solution concepts alternative to Nash Equilibria.

These concepts can be defined according to suitable extensions of the agents' rationality. In such a setting, a recent research direction has focused on sequential games [5,12], modeling the strategic behavior of agents who anticipate future strategic opportunities. More precisely, in the majority of equilibrium concepts, it is assumed that players simultaneously select their strategic choices. Even when dealing with the speed of convergence and best-response moves, the corresponding dynamics is actually the result of a myopic interaction among the players, in which each player merely selects the strategy being at the moment a good choice, without caring about the future evolutions of the game. In other words, the dynamics is not governed by farsighted strategic choices of the players. As a consequence, in sequential games, the notion of Subgame Perfect Equilibrium (SPE) [17] is preferred to that of Nash Equilibrium since it better captures the rationality of farsighted players. Hence, the loss of optimality due to the presence of selfish players in sequential games is measured by means of the Sequential Price of Anarchy (SeqPoA), that is, the price of anarchy of Subgame Perfect Equilibria.

*Related Work.* Isolation games were introduced in [19], where the authors give several results regarding the existence of pure Nash Equilibria and the convergence of better and best-response dynamics to any of such equilibria. In particular, they prove that, in any symmetric space, nearest-neighbor and total-distance

isolation games are potential games, which implies the existence of Nash Equilibria and convergence of any better-response dynamics. In the case of asymmetric spaces, however, deciding whether a given nearest-neighbor or total-distance isolation game possesses a Nash Equilibrium is NP-complete. Furthermore, in [4], the authors analyze the efficiency of pure Nash Equilibria in terms of the Price of Anarchy and Price of Stability, for the classes of isolation games introduced in [19], under the two social functions defined as the minimum utility per player and the sum of the players' utilities.

Very recently, Subgame Perfect Equilibria and their performance have been investigated in the context of auctions [13], cut, consensus, unrelated scheduling and fair cost sharing allocation games [5,12]. A desired and expected effect of Subgame Perfect Equilibria remarked in [5,12] is that the corresponding farsighted choices may reduce the induced Price of Anarchy. Other non-myopic extensions induced by more farsighted agents that take into account the long-term effects of their adopted strategies were considered in [3].

*Our Contribution.* Motivated by the above reasons, in this paper we study the performance of Subgame Perfect Equilibria in sequential isolation games. More precisely, we focus on nearest-neighbor and total-distance sequential isolation games, and we show upper and lower bounds on the Sequential Price of Anarchy under the two social functions defined as the minimum utility per player (MIN) and the sum of the players' utilities (SUM).

For the basic case of  $k = 2$  players, we prove that the Sequential Price of Anarchy is 1, i.e., every Subgame Perfect Equilibrium is an optimal solution, both for nearest-neighbor and total-distance sequential isolation games, under the two considered social functions. For the other cases ( $k \geq 3$  players), the obtained results are summarized in Table 1. For nearest-neighbor games with SUM social function and  $k \geq 4$  players, that is the case with an unbounded Sequential Price of Anarchy, we provide improved results for the special case of unweighted graphs by showing that the Sequential Price of Anarchy is between  $4\frac{k-3}{k}$  and 8.

**Table 1.** Sequential Price of Anarchy of isolation games for  $k \geq 3$  players

Social Function	Nearest-Neighbor	Total-Distance
MIN	2	between $\frac{k-1}{k-2}$ and 8
SUM	$\infty$ for $k \geq 4$ players between $\frac{3}{2}$ and 6 for $k = 3$ players	between $\frac{k(k-1)}{(k+1)(k-2)}$ and 3.765

*Paper Organization.* The next section contains the necessary definitions and notation. Sections 3 and 4 cover the study of the Sequential Price of Anarchy for nearest-neighbor and total-distance sequential isolation games, respectively. Finally, in Section 5, we address open problems and further research.

Due to space limitations, some proofs have been omitted.

## 2 Definitions and Notation

For an integer  $k > 0$ , we denote as  $[k]$  the set  $\{1, 2, \dots, k\}$ . We use boldface characters, such as  $\mathbf{s}$  and  $\mathbf{a}$ , to represent tuples. Given a  $k$ -tuple  $\mathbf{s}$ , for any  $i \in [k]$ , we denote as  $s_i$  the  $i$ th element of  $\mathbf{s}$ , that is, we assume that  $\mathbf{s} = (s_1, \dots, s_k)$ .

A **sequential game** is a triple  $G = ([k], A_{i \in [k]}, U_{i \in [k]})$ , where  $[k]$  is a set of  $k \geq 2$  *players* and, for any  $i \in [k]$ ,  $A_i$  is the *set of actions* for player  $i$  and  $U_i : \times_{j \in [k]} A_j \rightarrow \mathbb{R}_{>0}$  is her *utility function*. The game is played in  $k$  steps. At the  $i$ th step, player  $i$  observes the actions chosen by the first  $i - 1$  players and decides which action to take. Any function  $s_i : \times_{j \in [i-1]} A_j \rightarrow A_i$  is a *strategy* for player  $i$ . A *strategy profile*  $\mathbf{s}$  is a  $k$ -tuple of strategies, one for each player. Each strategy profile  $\mathbf{s}$  induces a unique *outcome*  $\mathbf{a}(\mathbf{s})$ , defined as  $a_1(\mathbf{s}) = s_1(\emptyset)$  and  $a_i(\mathbf{s}) = s_i(a_1(\mathbf{s}), \dots, a_{i-1}(\mathbf{s}))$  for any  $2 \leq i \leq k$ ; hence, the utility that each player  $i \in [k]$  gets as an outcome of the strategy profile  $\mathbf{s}$  is given by  $U_i(\mathbf{a}(\mathbf{s}))$ .

Each sequential game  $G$  can be represented by a tree  $T_G = (H, E)$ , where  $H = H_1 \cup \dots \cup H_{k+1}$ , with  $H_{k+1} = \times_{i \in [k]} A_i$ , and  $E$  is such that, for each  $i \in [k]$  and  $h \in H_i$ ,  $h$  has exactly  $|A_i|$  children corresponding to each of the possible action choices for player  $i$ . Hence, it holds  $|H_1| = 1$  and  $|H_{i+1}| = |H_i| \cdot |A_i|$  for each  $i \in [k]$ . Note that each strategy profile  $\mathbf{s}$  induces a unique path in  $T_G$  going from the root to the leave corresponding to  $\mathbf{a}(\mathbf{s})$ .

Each node  $h$  of  $T_G$  corresponds to a *history* of the game, that is, to the unique sequence of choices leading from the root to  $h$ . Given a node  $h \in H$ , the *descendants* of  $h$  are all the nodes belonging to the subtree of  $T_G$  rooted at  $h$ . The *subgame* of  $G$  rooted at node  $h$ , denoted as  $G_h$ , is the restriction of  $G$  to the descendants of  $h$ . The set of subgames of  $G$  consists of all of the subgames of  $G$  rooted at some node of  $G$ .

Given a strategy profile  $\mathbf{s}$ , a player  $i \in [k]$  and a strategy  $t$  for player  $i$ , we denote as  $\mathbf{s}_{-i} \diamond t$  the strategy profile obtained from  $\mathbf{s}$  when player  $i$  unilaterally changes her strategy from  $s_i$  to  $t$ .

A strategy profile  $\mathbf{s}$  is a **Nash Equilibrium** (NE) of  $G$  if, for each player  $i \in [k]$  and strategy  $t$  for player  $i$ , it holds  $U_i(\mathbf{a}(\mathbf{s})) \geq U_i(\mathbf{a}(\mathbf{s}_{-i} \diamond t))$ , while it is a **Subgame Perfect Equilibrium** (SPE) of  $G$  if, for any node  $h \in H$ , the restriction of  $\mathbf{s}$  to  $G_h$ , denoted as  $\mathbf{s}(G_h)$ , is a Nash Equilibrium of  $G_h$ .

We denote by  $\mathcal{SPE}(G)$  the set of Subgame Perfect Equilibria of game  $G$ . It is well-known that, for any sequential game  $G$ , it holds  $\mathcal{SPE}(G) \neq \emptyset$  and that such a set can be computed by backward induction on  $T_G$ . Moreover, the presence of tie-breaking situations in which the players can choose among equivalent actions possibly makes  $|\mathcal{SPE}(G)| > 1$ .

Given a sequential game  $G$  and a *social function*  $SF : \times_{i \in [k]} A_i \mapsto \mathbb{R}_{>0}$ , let  $\mathbf{o}_{SF}(G) \in \times_{i \in [k]} A_i$  be an outcome maximizing  $SF$ . The **Sequential Price of Anarchy** (SeqPoA) of  $G$  is defined as  $\text{SeqPoA}_{SF}(G) = \max_{\mathbf{s} \in \mathcal{SPE}(G)} \frac{SF(\mathbf{o}_{SF}(G))}{SF(\mathbf{a}(\mathbf{s}))}$ .

For an undirected connected graph  $G = (V, E, w)$  with  $w : E \rightarrow \mathbb{R}_{>0}$  and two nodes  $u, v \in V$ , let  $d(u, v)$  denote the distance between  $u$  and  $v$  in  $G$ , that is, the length of the shortest  $(u, v)$ -path in  $G$ . Given an undirected connected graph  $G = (V, E, w)$  and an integer  $k \geq 2$ , a **sequential isolation game**  $I = (G, k)$  is a sequential game such that, for any  $i \in [k]$ ,  $A_i = V$ , i.e., the

action of each player is to choose one of the  $n = |V|$  nodes of  $G$ . Given an action profile  $\mathbf{a}$ , for any player  $i \in [k]$ , define the *distance vector* of  $i$  in  $\mathbf{a}$  as  $\mathbf{b}^i(\mathbf{a}) = (d(s_i, s_1), \dots, d(s_i, s_{i-1}), d(s_i, s_{i+1}), \dots, d(s_i, s_k))$ . The utility  $U_i(\mathbf{a})$  that player  $i$  gets in the action profile  $\mathbf{a}$  can be defined in several ways on the basis of the distance vector  $\mathbf{b}^i(\mathbf{a})$ . We consider the following two cases:

- *nearest-neighbor sequential isolation games*, where, for any  $i \in [k]$ ,  $U_i(\mathbf{a}) = \min_{j \in [k-1]} \{b_j^i(\mathbf{a})\}$ , that is, the utility of a player is given by the distance from her nearest neighbor in  $\mathbf{a}$ ;
- *total-distance sequential isolation games*, where, for any  $i \in [k]$ ,  $U_i(\mathbf{a}) = \sum_{j \in [k-1]} b_j^i(\mathbf{a})$ , that is, the utility of a player is given by the sum of her distances from all the other players in  $\mathbf{a}$ .

We study the SeqPoA of nearest-neighbor and total-distance sequential isolation games under the two standard social functions adopted in the literature, namely, the minimum utility per player  $\text{MIN}(\mathbf{a}) = \min_{i \in [k]} \{U_i(\mathbf{a})\}$  and the sum of the utilities of all the players  $\text{SUM}(\mathbf{a}) = \sum_{i \in [k]} U_i(\mathbf{a})$ .

### 3 Nearest-Neighbor Sequential Isolation Games

Before presenting our technical results, we revise the definition and the properties of an  $i$ -center in undirected weighted connected graphs.

An  $i$ -center for an undirected weighted connected graph  $G = (V, E, w)$  is a subset of nodes  $\mathcal{C} \subseteq V$  such that  $|\mathcal{C}| \leq i$ . The measure of an  $i$ -center  $\mathcal{C}$  is defined as  $m(\mathcal{C}) = \max_{v \in V} \{\min_{u \in \mathcal{C}} \{d(u, v)\}\}$ . An optimal  $i$ -center is an  $i$ -center of minimum measure. We denote by  $\mathcal{C}_i^* = \{c_1^*, \dots, c_{|\mathcal{C}_i^*|}^*\}$  an optimal  $i$ -center for  $G$ . Note that, by definition, it holds  $m(\mathcal{C}_{i+1}^*) \leq m(\mathcal{C}_i^*)$ . Given an  $i$ -center  $\mathcal{C}$  and a node  $v \in V$ , let  $f_{\mathcal{C}}(v) = \operatorname{argmin}_{u \in \mathcal{C}} \{d(u, v)\}$  be the node at minimum distance from  $v$  among the ones belonging to  $\mathcal{C}$ , breaking ties arbitrarily. Each  $i$ -center  $\mathcal{C} = (c_1, \dots, c_{|\mathcal{C}|})$  induces a partition of  $V$  into  $|\mathcal{C}|$  clusters  $C_1, \dots, C_{|\mathcal{C}|}$ , where  $C_i = \{v \in V : f_{\mathcal{C}}(v) = c_i\}$ . Let  $W = \max_{\{u, v\} \in E} \{w(\{u, v\})\}$  be the weight of the most heavy edge in  $G$ . For an integer  $i \geq 2$ , any  $i$ -center  $\mathcal{C}$  for  $G$  satisfies the following two properties:

*Property 1.* For any  $i \in [|\mathcal{C}|]$  and two nodes  $u, v \in C_i$ , it holds  $d(u, v) \leq 2 \cdot m(\mathcal{C})$ .

*Property 2.* If  $|\mathcal{C}| > 1$ , for any node  $u \in \mathcal{C}$ , there exists another node  $v \in \mathcal{C}$ , with  $u \neq v$ , such that  $d(u, v) \leq 2 \cdot m(\mathcal{C}) + W$ .

Throughout this section, we shall denote by  $I$  an instance of nearest-neighbor isolation games. The following lemma provides an upper bound to the utility realized by each player  $i \in [k]$  in any SPE.

**Lemma 1.** *For any  $s \in \mathcal{SPE}(I)$  and  $i \in [k]$ , it holds  $U_i(\mathbf{a}(s)) \geq m(\mathcal{C}_{k-1}^*)$ .*

*Proof.* For any instance  $I$ , consider the choice of player  $i > 1$ . We claim that there exists an action  $v \in V$  for player  $i$  such that  $\min_{j \in [i-1]} \{d(a_j, v)\} \geq m(\mathcal{C}_{i-1}^*)$ . In

fact, if by contradiction  $\min_{j \in [i-1]} \{d(a_j, v)\} < m(\mathcal{C}_{i-1}^*)$  for every  $v \in V$ , it follows that the set  $X = \bigcup_{j \in [i-1]} \{a_j\}$  is an  $(i-1)$ -center with  $m(X) < m(\mathcal{C}_{i-1}^*)$ , thus contradicting the optimality of  $\mathcal{C}_{i-1}^*$ .

We now show by backward induction that, for each player  $i \in [k]$ , it holds  $U_i(\mathbf{a}(\mathbf{s})) \geq m(\mathcal{C}_{k-1}^*)$ . For  $i = k$ , since there is an action  $v \in V$  such that  $\min_{j \in [k-1]} \{d(a_j, v)\} \geq m(\mathcal{C}_{k-1}^*)$  and player  $k$  aims at maximizing her utility, it follows that, in any SPE  $\mathbf{s}$ , it holds  $U_k(\mathbf{a}(\mathbf{s})) \geq m(\mathcal{C}_{k-1}^*)$ . For  $1 \leq i < k$ , we can suppose, for the sake of induction, that, for any action  $v \in V$  eventually chosen by player  $i$ , it holds  $d(a_j(\mathbf{s}), v) \geq m(\mathcal{C}_{k-1}^*)$  for any  $j = i+1, \dots, k$ . Moreover, we know that there exists an action  $v \in V$  for player  $i$  such that  $\min_{j \in [i-1]} \{d(a_j(\mathbf{s}), v)\} \geq m(\mathcal{C}_{i-1}^*) \geq m(\mathcal{C}_{k-1}^*)$ . Hence, there exists an action  $v \in V$  for player  $i$  such that  $\min_{j \in [k]} \{d(a_j(\mathbf{s}), v)\} \geq m(\mathcal{C}_{k-1}^*)$ . Since player  $i$  aims at maximizing her utility, it follows that, in any SPE  $\mathbf{s}$ , it holds  $U_i(\mathbf{a}(\mathbf{s})) \geq m(\mathcal{C}_{k-1}^*)$  for any  $i \in [k]$ .  $\square$

By exploiting the above lemma, we can obtain a significant upper bound on the SeqPoA for the social function MIN.

**Theorem 1.** *For any instance  $I$ , it holds  $\text{SeqPoA}_{\text{MIN}}(I) \leq 2$ .*

*Proof.* Consider the partition of  $V$  induced by  $\mathcal{C}_{k-1}^*$ . Since  $V$  is partitioned into  $k-1$  clusters, for any action profile  $\mathbf{a}$ , there must exist two players whose choices belong to the same cluster. That is, there must be a vertex  $u \in \mathcal{C}_{k-1}^*$  and two players  $i$  and  $j$  such that  $d(u, a_i) \leq m(\mathcal{C}_{k-1}^*)$  and  $d(u, a_j) \leq m(\mathcal{C}_{k-1}^*)$ . By the definition of distance, it follows that  $d(a_i, a_j) \leq 2 \cdot m(\mathcal{C}_{k-1}^*)$  which implies that  $\text{MIN}(\mathbf{o}_{\text{MIN}}(I)) \leq 2 \cdot m(\mathcal{C}_{k-1}^*)$ . The thesis then follows by Lemma 1.  $\square$

We proceed by showing that the given upper bound is tight for any number of players  $k \geq 3$ .

**Theorem 2.** *For any number of players  $k \geq 3$ , there exists an instance  $I_k$  of nearest-neighbor sequential isolation games such that  $\text{SeqPoA}_{\text{MIN}}(I_k) = 2$ .*

*Proof.* For  $k = 3$ , let  $I_3 = (G, 3)$  be the instance of nearest-neighbor sequential isolation games yielded by the case in which  $G$  is the unweighted path  $\langle v_1, v_2, \dots, v_5 \rangle$ . We show that there exists an SPE  $\mathbf{s}$  such that  $\mathbf{a}(\mathbf{s}) = (v_2, v_4, v_5)$  for which  $\text{MIN}(\mathbf{a}(\mathbf{s})) = 1$ . Since the action profile  $\mathbf{o}_{\text{MIN}}(I_3) = (v_1, v_3, v_5)$  yields  $\text{MIN}(\mathbf{o}_{\text{MIN}}(I_3)) = 2$ , it then follows that  $\text{SeqPoA}_{\text{MIN}}(I_3) = 2$ . First of all, it is not difficult to see that, for any SPE  $\mathbf{s}'$  of  $I_3$ , it must be  $U_i(\mathbf{a}(\mathbf{s}')) \leq 2$  for any  $i \in [3]$ . Hence, since  $U_1(\mathbf{a}(\mathbf{s})) = 2$ , it follows that player 1 is happy with her choice in  $\mathbf{s}$ . Player 2 is getting a utility of 1 in  $\mathbf{a}(\mathbf{s})$ . The only deviation that does not immediately give her a utility of 1 is to choose node  $v_5$ . In such a case, player 3 cannot get more than 1 and so we can correctly assume that she chooses node  $v_4$  which again gives a utility of 1 to player 2. So player 2 cannot improve her utility by deviating from  $\mathbf{s}$ . Finally, player 3 is getting a utility of 1 in  $\mathbf{a}(\mathbf{s})$ . Since she cannot get more than 1 once that the first two players have chosen nodes  $v_2$  and  $v_4$ , it follows that player 3 cannot improve her utility by deviating from  $\mathbf{s}$  and this shows that  $\mathbf{s}$  is an SPE.

For any integer  $k \geq 4$ , let  $I_k = (G_k, k)$  be the instance of nearest-neighbor sequential isolation games yielded by the graph  $G_k$  depicted in Figure 1. In an optimal action profile  $\mathbf{o}_{\text{MIN}}(I_k)$ , each player  $i \in [k-1]$  chooses node  $v_i$  and player  $k$  chooses node  $r$  which results in  $\text{MIN}(\mathbf{o}_{\text{MIN}}(I_k)) = 2$ . We claim that the strategy profile  $\mathbf{s}$  for which  $\mathbf{a}(\mathbf{s})$  is such that each player  $i \in [k-2]$  chooses node  $u_i$ , player  $k-1$  chooses node  $v_{k-2}$  and player  $k$  chooses  $v_{k-1}$  is an SPE. Since it holds  $\text{MIN}(\mathbf{a}(\mathbf{s})) = 1$ , the thesis follows.

Note that, for any SPE  $\mathbf{s}'$ , it holds  $U_i(\mathbf{a}(\mathbf{s}')) \leq 2$  for any player  $i \in [k]$ . In fact, assume by way of contradiction that there exists an SPE  $\mathbf{s}'$  such that  $U_i(\mathbf{a}(\mathbf{s}')) > 2$  for a player  $i \in [k]$ . By the topology of  $G_k$ , this can only happen if  $a_i(\mathbf{s}') = v_\ell$ , for some  $\ell \in [k-1]$  and  $a_j(\mathbf{s}') \notin \{r, u_\ell\}$  for any  $j \in [k]$ . This implies that there exists a player  $j \in [k]$  such that  $U_j(\mathbf{a}(\mathbf{s}')) = 1$ . Since for any choice the remaining players can make at equilibrium (i.e., never selecting nodes already taken by other players) it holds that the utility that player  $j$  gets in the subgame of  $I_k$  obtained when player  $j$  deviates to node  $r$  is at least 1, we can correctly assume that player  $j$  breaks tie in favor of action  $r$ . In this case, the utility of player  $i$  drops to 2 and the claim follows.

We now complete the proof by showing that  $\mathbf{s}$  is an SPE. Note that, since  $U_i(\mathbf{a}(\mathbf{s})) = 2$  for any  $i \in [k-3]$  and no player can achieve a utility better than 2 in any SPE, it follows that the first  $k-3$  players are happy with their choices. When player  $k-2$  chooses  $u_{k-2}$ , the remaining two players, no matter what they choose, cannot get a utility better than 1; hence, since  $U_{k-1}(\mathbf{a}(\mathbf{s})) = U_k(\mathbf{a}(\mathbf{s})) = 1$ , it follows that players  $k-1$  and  $k$  are happy with their choices too. Hence, it remains to show that player  $k-2$ , for which  $U_{k-2}(\mathbf{a}(\mathbf{s})) = 1$ , does not want to deviate from her strategy in  $\mathbf{s}$ . Once given the choices of the first  $k-3$  players, the only choices (different from  $u_{k-2}$ ) which does not immediately gives a utility of 1 to player  $k-2$  are either  $v_{k-2}$  or  $v_{k-1}$ . Assume that player  $k-2$  chooses  $v_{k-2}$  (the other case can be treated symmetrically). It is easy to see now that the strategy profile generating the outcome in which the remaining two players choose  $v_{k-1}$  and  $u_{k-2}$  is an SPE such that the utility of player  $k-2$  is again 1. Hence, it follows that  $\mathbf{s}$  is an SPE.  $\square$

For the leftover case of  $k = 2$ , the following result can be easily proved.

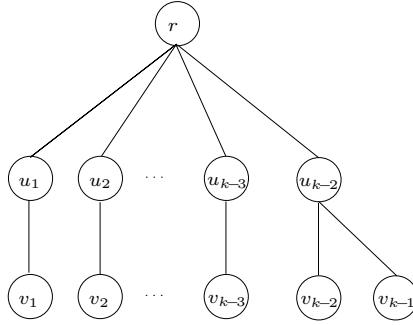
**Theorem 3.** *For any instance  $I = (G, 2)$ , it holds  $\text{SeqPoA}_{\text{MIN}}(I) = \text{SeqPoA}_{\text{SUM}}(I) = 1$ .*

For the social function **SUM**, we can show the following negative result for any number of players  $k \geq 4$ .

**Theorem 4.** *For any number of players  $k \geq 4$ , there exists an instance  $I_k$  such that  $\text{SeqPoA}_{\text{SUM}}(I_k)$  is unbounded.*

For the leftover case of  $k = 3$ , we can show the following bounds.

**Theorem 5.** *For any instance  $I_3 = (G, 3)$ , it holds  $\text{SeqPoA}_{\text{SUM}}(I) \leq 6$ . Moreover, there exists an instance  $I_3 = (G, 3)$  for which  $\text{SeqPoA}_{\text{SUM}}(I) = 3/2$ .*



**Fig. 1.** An unweighted graph yielding a nearest-neighbor sequential isolation game whose SeqPoA for the social function MIN is 2

Because of the negative result shown in Theorem 4, we restrict our attention to instances in which  $G$  is unweighted, that is,  $w(\{u, v\}) = 1$  for any  $\{u, v\} \in E$ . In such a case, we can prove the following upper bound.

**Theorem 6.** *For any instance  $I = (G, k)$  such that  $G$  is unweighted, it holds  $\text{SeqPoA}_{\text{SUM}}(I) \leq 8$ .*

*Proof.* For any action profile  $\mathbf{a} = (a_1, \dots, a_k)$ , let  $A = \bigcup_{i \in [k]} \{a_i\}$  be the set of actions chosen by at least one player in  $\mathbf{a}$ . For any  $i \in [\lvert \mathcal{C}_{k-1}^* \rvert]$ , denote  $D_i = \mathcal{C}_i^* \cap A$ . We say that a cluster  $D_i$  is crowded if  $|D_i| \geq 2$ . Let  $\mathcal{CR}$  be the union of all the crowded clusters. Let  $T$  denote a spanning tree connecting all the nodes in  $\mathcal{C}_{k-1}^*$  defined on the metric closure of  $G$  and consider the ordering of the nodes in  $\mathcal{C}_{k-1}^*$  induced by a depth-first search on  $T$ . For a non-crowded cluster  $D_i$ , let  $\text{succ}(i)$  denote  $j \bmod |\mathcal{C}_{k-1}^*|$ , where  $j$  is the minimum index  $j > i$  such that  $D_j \bmod |\mathcal{C}_{k-1}^*|$  is crowded. We can bound  $\text{SUM}(\mathbf{a})$  as follows.

$$\begin{aligned} \text{SUM}(\mathbf{a}) &\leq \sum_{i \in [k]: a_i \in \mathcal{CR}} 2m(\mathcal{C}_{k-1}^*) + \sum_{i \in [k]: a_i \notin \mathcal{CR}} \left( 2m(\mathcal{C}_{k-1}^*) + d(f_{\mathcal{C}_{k-1}^*}(a_i), c_{\text{succ}(i)}^*) \right) \\ &\leq \sum_{i \in [k]} 2m(\mathcal{C}_{k-1}^*) + 2(k-2)(2m(\mathcal{C}_{k-1}^*) + 1) \\ &\leq 8km(\mathcal{C}_{k-1}^*), \end{aligned}$$

where the second inequality follows from the fact that, in the worst-case all the edges in  $T$  are visited at most twice,  $T$  has exactly  $k-2$  edges, and the cost of each edge is at most  $2 \cdot m(\mathcal{C}_{k-1}^*) + 1$  by Property 2; while the last inequality follows from the fact that  $m(\mathcal{C}_{k-1}^*) \geq 1$ . The thesis then follows by Lemma 1.  $\square$

The following lower bound, approaching 4 as  $k$  tends to infinity, holds.

**Theorem 7.** *For any number of players  $k \geq 4$ , there exists an instance  $I_k = (G_k, k)$  with  $G_k$  being an unweighted graph, such that  $\text{SeqPoA}_{\text{SUM}}(I_k) = 4^{\frac{k-3}{k}}$ .*

## 4 Total-Distance Sequential Isolation Games

Throughout this section, we shall denote by  $I$  an instance of total-distance sequential isolation game and with  $D$  the diameter of  $G$ , that is, the maximum distance between any two nodes.

The following lemma provides an upper bound to the utility realized by each player  $i \in [k]$  at any SPE when  $k > 2$ .

**Lemma 2.** *For any  $\mathbf{s} \in \mathcal{SPE}(I)$  and  $i \in [k]$ , with  $k > 2$ , it holds  $U_i(\mathbf{a}(\mathbf{s})) \geq \max\left\{\frac{(i-1)D}{2}; \frac{(k-1)D}{8}\right\}$ .*

*Proof.* We first prove that, for any  $i \in [k]$ , it holds  $U_i(\mathbf{a}(\mathbf{s})) \geq \frac{(i-1)D}{2}$ . To this aim, let  $u, v \in V$  be two nodes such that  $d(u, v) = D$ . Consider the choice of player  $i$  and note that, for any  $j \in [i-1]$ , it holds  $d(u, a_j(\mathbf{s})) + d(a_j(\mathbf{s}), v) \geq D$  by the definition of distance. Hence, by summing up for all  $j \in [i-1]$ , it follows  $\sum_{j \in [i-1]} (d(u, a_j(\mathbf{s})) + d(a_j(\mathbf{s}), v)) \geq (i-1)D$ . By a simple averaging argument, it follows that one of the two choices  $u, v$  guarantees a utility of at least  $(i-1)D/2$  to player  $i$ .

We now proceed by proving that, for any  $i \in [k]$ , it holds  $U_i(\mathbf{a}(\mathbf{s})) \geq \frac{(k-1)D}{8}$ . When  $i \geq \frac{k-1}{4} + 1$ , since  $U_i(\mathbf{a}(\mathbf{s})) \geq \frac{(i-1)D}{2}$ , it follows that  $U_i(\mathbf{a}(\mathbf{s})) \geq \frac{(i-1)D}{2} \geq \frac{(k-1)D}{8}$ . Hence, for the sake of contradiction, we can assume that  $i < \frac{k-1}{4} + 1$  and  $U_i(\mathbf{a}(\mathbf{s})) < \frac{(k-1)D}{8}$ . For any  $j$  such that  $i+1 \leq j \leq k$ , it holds  $U_j(\mathbf{a}(\mathbf{s})) \leq U_i(\mathbf{a}(\mathbf{s})) + (k-2)d(a_i(\mathbf{s}), a_j(\mathbf{s}))$  which implies  $U_i(\mathbf{a}(\mathbf{s})) \geq U_j(\mathbf{a}(\mathbf{s})) - (k-2)d(a_i(\mathbf{s}), a_j(\mathbf{s})) \geq \frac{(j-1)D}{2} - (k-2)d(a_i(\mathbf{s}), a_j(\mathbf{s}))$ . Note that, when  $d(a_i(\mathbf{s}), a_j(\mathbf{s})) \leq \frac{\frac{(j-1)D}{2} - \frac{(k-1)D}{8}}{k-2}$ , it follows that  $U_i(\mathbf{a}(\mathbf{s})) \geq \frac{(k-1)D}{8}$ . Hence, we can further assume that  $d(a_i(\mathbf{s}), a_j(\mathbf{s})) > \frac{\frac{(j-1)D}{2} - \frac{(k-1)D}{8}}{k-2}$  for any  $j$  such that  $i+1 \leq j \leq k$ . As a consequence, we obtain that

$$\begin{aligned} U_i(\mathbf{a}(\mathbf{s})) &> \frac{D}{k-2} \sum_{j=\frac{k-1}{4}+1}^k \left( \frac{j-1}{2} - \frac{k-1}{8} \right) \\ &= \frac{D}{4(k-2)} \left( \left[ \frac{3k+1}{4} \right]^2 + \left[ \frac{3k+1}{4} \right] \right). \end{aligned}$$

In order to conclude the proof, we need to show that  $\frac{D}{4(k-2)} \left( \left[ \frac{3k+1}{4} \right]^2 + \left[ \frac{3k+1}{4} \right] \right) \geq \frac{(k-1)D}{8}$ . For the cases of  $k = 3, 4$ , this can be easily checked by inspection. In order to show it for the remaining values of  $k$ , note that  $\frac{D}{4(k-2)} \left( \left[ \frac{3k+1}{4} \right]^2 + \left[ \frac{3k+1}{4} \right] \right) > \frac{D}{4(k-2)} \left( \left( \frac{3k+1}{4} - 1 \right)^2 + \frac{3k+1}{4} - 1 \right) \geq \frac{(k-1)D}{8}$  for any  $k \geq 5$ .  $\square$

By exploiting the above lemma, we can obtain the following upper bounds on the SeqPoA of total-distance sequential isolation games under both social functions MIN and SUM.

**Theorem 8.** For any instance  $I$  with  $k > 2$ , it holds  $\text{SeqPoA}_{\text{MIN}}(I) \leq 8$ .

*Proof.* For an instance  $I$ , it clearly holds that  $\text{MIN}(\mathbf{o}_{\text{MIN}}(I)) \leq (k-1)D$ . Thus, the claim immediately follows from Lemma 2.  $\square$

**Theorem 9.** For any instance  $I$ , it holds  $\text{SeqPoA}_{\text{SUM}}(I) \leq \frac{64}{17} \approx 3.765$ .

*Proof.* Let  $\mathbf{s}$  be an SPE for game  $I$ . By Lemma 2, it follows that

$$\begin{aligned} \text{SUM}(\mathbf{a}(\mathbf{s})) &= \sum_{i \in [k]} U_i(\mathbf{a}(\mathbf{s})) \\ &= \sum_{i=1}^{\lceil \frac{k-1}{4} \rceil} U_i(\mathbf{a}(\mathbf{s})) + \sum_{i=\lceil \frac{k-1}{4} \rceil+1}^k U_i(\mathbf{a}(\mathbf{s})) \\ &\geq \sum_{i=1}^{\lceil \frac{k-1}{4} \rceil} \frac{(k-1)D}{8} + \sum_{i=\lceil \frac{k-1}{4} \rceil+1}^k \frac{(i-1)D}{2} \\ &= \left\lceil \frac{k-1}{4} \right\rceil \frac{(k-1)D}{8} + \frac{k(k-1)D}{4} - \left\lceil \frac{k-1}{4} \right\rceil \left( \left\lceil \frac{k-1}{4} \right\rceil - 1 \right) \frac{D}{4} \\ &= \frac{D}{4} \left( k(k-1) - \left\lceil \frac{k-1}{4} \right\rceil^2 + \left\lceil \frac{k-1}{4} \right\rceil \frac{k+1}{2} \right) \\ &= \frac{D}{4} \left( k(k-1) + \left\lceil \frac{k-1}{4} \right\rceil \left( \frac{k+1}{2} - \left\lceil \frac{k-1}{4} \right\rceil \right) \right) \\ &> \frac{D}{4} \left( k(k-1) + \frac{k-1}{4} \left( \frac{k+1}{2} - \frac{k+2}{4} \right) \right) \\ &= \frac{17k(k-1)D}{64}. \end{aligned}$$

Since it clearly holds that  $\text{SUM}(\mathbf{o}_{\text{SUM}}(I)) \leq k(k-1)D$ , the claim follows.  $\square$

The following lower bounds hold for any number of players  $k \geq 3$ .

**Theorem 10.** For any number of players  $k \geq 3$ , there exists an instance  $I_k = (G_k, k)$  such that  $\text{SeqPoA}_{\text{MIN}}(I_k) = \frac{k-1}{k-2}$  and  $\text{SeqPoA}_{\text{SUM}}(I_k) = \frac{k(k-1)}{(k+1)(k-2)}$ .

Unfortunately, the quality of the derived lower bounds decreases when the number of players increases, as they tend to 1 when  $k$  goes to infinity. This behavior, in some sense, meets the perception that, when there is an arbitrarily high number of players in the game, it is unlikely that, when applying a farsighted rationality, they will not end up in an action profile in which all but a negligible part of them are distanced themselves for as much as possible. Hence, disproving this belief by designing constant lower bounds for any number of players or determining better upper bounds tending to 1 as the number of players goes to infinity remains a major open problem.

For the leftover case of  $k = 2$ , we can show the following result.

**Theorem 11.** For any instance  $I = (G, 2)$ , it holds  $\text{SeqPoA}_{\text{MIN}}(I) = \text{SeqPoA}_{\text{SUM}}(I) = 1$ .

## 5 Conclusions

We have studied the performance of Subgame Perfect Equilibria in the context of sequential isolation games. More precisely, we have focused on two classes of sequential isolation games, namely nearest-neighbor and total-distance sequential isolation games, and we have bounded their Sequential Price of Anarchy under both the minimum utility social function and the sum of the players' utilities social function.

Several questions are still open. Besides tightening the upper and lower bounds on the Sequential Price of Anarchy, it would be nice to investigate other more general classes of sequential isolation games, in which for instance the utility of players is defined as follows: For any player  $i$ , each strategy profile  $S$  yields a vector  $\mathbf{f}^i(S) \in \mathbb{R}_{\geq 0}^{k-1}$  such that the  $j$ -th component of  $\mathbf{f}^i(S)$  is the distance between the location chosen by player  $i$  and the one chosen by her  $j$ -th nearest player; the utility of player  $i$  can thus be defined as any convex combination of the elements of  $\mathbf{f}^i(S)$ .

Our results are quite interesting and difficult to be interpreted, as they show that a farsighted behavior, at least in the worst case, does not improve the performance of the obtained equilibria with respect to a myopic classical Nash dynamics (see Table 2 for a comparison with the upper bounds on the “classical” Price of Anarchy obtained in [4]). This suggests the adoption and investigation of reasonable variants of SPE exhibiting better performances and removing some excessively strong assumptions that can limit their applicability, like the fact that every player has full information of the actions selected by every other players in each possible state of the game. To this respect, considering SPE in the framework of [2], in which graphical games (i.e., games in which each player is only aware of the existence of the players she is connected to in a given social knowledge graph) are analyzed, constitutes an interesting research direction.

**Table 2.** Upper bounds on the Price of Anarchy of isolation games, with  $k$  being the number of players [4]

Social Function	Nearest-Neighbor	Total-Distance
MIN	2	$2^{\frac{k+1}{k-1}}$
SUM	$\infty$	2

## References

1. Anshelevich, E., Dasgupta, A., Tardos, É., Wexler, T.: Near-Optimal Network Design with Selfish Agents. In: Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), pp. 511–520. ACM Press (2003)

2. Bilò, V., Fanelli, A., Flammini, M., Moscardelli, L.: When Ignorance Helps: Graphical Multicast Cost Sharing Games. *Theoretical Computer Science* 411(3), 660–671 (2010)
3. Bilò, V., Flammini, M.: Extending the Notion of Rationality of Selfish Agents: Second Order Nash Equilibria. *Theoretical Computer Science* 412(22), 2296–2311 (2011)
4. Bilò, V., Flammini, M., Monaco, G., Moscardelli, L.: On the Performances of Nash Equilibria in Isolation Games. *Journal of Combinatorial Optimization* 22, 378–397 (2011)
5. Bilò, V., Flammini, M., Monaco, G., Moscardelli, L.: Some Anomalies of Farsighted Strategic Behavior. In: Proceedings of the 10th Workshop on Approximation and Online Algorithms (WAOA), Ljubljana, Slovenia, September 13–14 (2012)
6. Cheong, O., Har-Peled, S., Linial, N., Matousek, J.: The One-Round Voronoi Game. *Discrete and Computational Geometry* 31, 125–138 (2004)
7. Dürr, C., Thang, N.K.: Nash Equilibria in Voronoi Games on Graphs. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) *ESA 2007. LNCS*, vol. 4698, pp. 17–28. Springer, Heidelberg (2007)
8. Eiselt, H.A., Laporte, G., Thisse, J.F.: Competitive Location Models: A Framework and Bibliography. *Transportation Science* 27(1), 44–54 (1993)
9. Hotelling, H.: Stability in Competition. *Computational Geometry: Theory and Applications* 39(153), 41–57 (1929)
10. Jain, A.K., Murty, M.N., Flynn, P.J.: Data Clustering: A Review. *ACM Computing Surveys* 31(3) (1999)
11. Koutsoupias, E., Papadimitriou, C.: Worst-Case Equilibria. In: Meinel, C., Tison, S. (eds.) *STACS 1999. LNCS*, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
12. Paes Leme, R., Syrgkanis, V., Tardos, É.: The Curse of Simultaneity. In: *Proceedings of Innovations in Theoretical Computer Science (ITCS)*, pp. 60–67. ACM Press (2012)
13. Paes Leme, R., Syrgkanis, V., Tardos, É.: Sequential Auctions and Externalities. In: *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 869–886. SIAM (2012)
14. Mavronikolas, M., Monien, B., Papadopoulou, V.G., Schoppmann, F.: Voronoi Games on Cycle Graphs. In: Ochmański, E., Tyszkiewicz, J. (eds.) *MFCS 2008. LNCS*, vol. 5162, pp. 503–514. Springer, Heidelberg (2008)
15. Nash, J.: Non-Cooperative Games. *Annals of Mathematics* 54(2), 286–295 (1951)
16. Nisan, N., Roughgarden, T., Tardos, É., Vazirani, V.: *Algorithmic Game Theory*. Cambridge University Press, Cambridge (2007)
17. Osborne, M.J., Rubinstein, A.: *A Course in Game Theory*. MIT Press (1994)
18. Teng, S.H.: Low Energy and Mutually Distant Sampling. *Journal of Algorithms* 30(1), 42–67 (1999)
19. Zhao, Y., Chen, W., Teng, S.-H.: The Isolation Game: A Game of Distances. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) *ISAAC 2008. LNCS*, vol. 5369, pp. 148–158. Springer, Heidelberg (2008)

# Social Exchange Networks with Distant Bargaining\*

Konstantinos Georgiou<sup>1</sup>, George Karakostas<sup>2</sup>, Jochen Könemann<sup>1</sup>,  
and Zuzanna Stamirowska<sup>3</sup>

<sup>1</sup> Department of Combinatorics & Optimization, University of Waterloo  
[{k2georgiou,jochen}@math.uwaterloo.ca](mailto:{k2georgiou,jochen}@math.uwaterloo.ca)

<sup>2</sup> Department of Computing & Software, McMaster University  
[karakos@mcmaster.ca](mailto:karakos@mcmaster.ca)

<sup>3</sup> École Polytechnique, Paris, France  
[zuzanna.stamirowska@sciences-po.org](mailto:zuzanna.stamirowska@sciences-po.org)

**Abstract.** *Network bargaining* is a natural extension of the classical, 2-player *Nash bargaining* solution to the network setting. Here one is given an exchange network  $G$  connecting a set of players  $V$  in which edges correspond to potential contracts between their endpoints. In the standard model, a player may engage in at most one contract, and feasible outcomes therefore correspond to matchings in the underlying graph. Kleinberg and Tardos [STOC'08] recently proposed this model, and introduced the concepts of *stability* and *balance* for feasible outcomes. The authors characterized the class of instances that admit such solutions, and presented a polynomial-time algorithm to compute them.

In this paper, we generalize the work of Kleinberg and Tardos by allowing agents to engage into more complex contracts that span more than two agents. We provide suitable generalizations of the above stability and balance notions, and show that many of the previously known results for the matching case extend to our new setting. In particular, we can show that a given instance admits a stable outcome only if it also admits a balanced one. Like Batani et al. [ICALP'10] we exploit connections to cooperative games. We fully characterize the core of these games, and show that checking its non-emptiness is NP-complete. On the other hand, we provide efficient algorithms to compute core elements for several special cases of the problem, making use of compact linear programming formulations.

## 1 Introduction

The study of bargaining has been a central theme in economics and sociology, since it constitutes a basic activity in any human society. The most basic bargaining model is that of two agents A and B that negotiate how to divide a good of a certain value (say, 1) amongst themselves, while at the same time each has an *outside option* of value  $\alpha$  and  $\beta$  respectively. The famous Nash bargaining solution [12] postulates that in an *equitable* outcome, each player should receive her outside option, and that the surplus  $s = 1 - \alpha - \beta$  is to be split evenly between A and B.

---

\* This work was initiated in the International Problem Solving Workshop, held in July 16-20, 2012, and organized as part of the MITACS International Focus Period “Advances in Network Analysis and its Applications”. We would like to thank MITACS for this great opportunity.

More recently, Kleinberg and Tardos [10] proposed the following natural *network* extension of this game. Here, the set of players corresponds to the vertices of an undirected graph  $G = (V, E)$ ; each edge  $ij \in E$  represents a potential *contract* between players  $i$  and  $j$  of value  $w_{ij} \geq 0$ . In Kleinberg and Tardos' model, players are restricted to form contracts with at most one of their neighbours. Outcomes of the *network bargaining* game are therefore given by a matching  $M \subseteq E$ , and an *allocation*  $x \in \mathbb{R}_+^V$  such that  $x_i + x_j = w_{ij}$  for all  $ij \in M$ , and  $x_i = 0$  if  $i$  is not incident to an edge of  $M$ .

Unlike in the non-network bargaining game, the outside option  $\alpha_i$  of player  $i$  is not a given parameter but rather implicitly determined by the network neighbourhood of  $i$ . Specifically, in an outcome  $(M, x)$ , player  $i$ 's outside option is defined as  $\alpha_i = \max\{w_{ij} - x_j : ij \in \delta(i) \setminus M\}$ , where  $\delta(i)$  is the set of edges incident to  $i$ . An outcome  $(M, x)$  is then called *stable* if  $x_i + x_j \geq w_{ij}$  for all edges  $ij \in E$ , and it is *balanced* if in addition, the value of the edges in  $M$  is split according to Nash's bargaining solution; i.e., for an edge  $ij$ ,  $x_i - \alpha_i = x_j - \alpha_j$ . Kleinberg and Tardos provide a characterization of the class of graphs that admit balanced outcomes, and present a combinatorial algorithm that computes one if it exists.

Bateni et al. [1] recently exhibited a close link between the study of network bargaining and that of *matching games* in cooperative game theory. The authors showed that stable outcomes for an instance of network bargaining correspond to allocations in the *core* of the underlying matching game. Moreover, balanced outcomes correspond to *prekernel* allocations. As a corollary, this implies that an algorithm by Faigle et al. [7] gives an alternate method to obtain balanced outcomes in a network bargaining game. Bateni et al. also extended the work of [10] to bipartite graphs in which the agents of one side are allowed to engage in more than one contract.

Matching games have indeed been studied extensively in the game theory community since the early 70s, when Shapley and Shubik investigated the core of the class of bipartite matching games, so called *assignment games*, in their seminal paper [14]. Granot and Granot [8] also study the core of the assignment game; the authors show that it contains many points, some of which may not be desirable ways to share revenue. The authors propose to focus on the intersection of core and prekernel instead, and provide sufficient and necessary conditions for the former to be contained in the latter. Deng et al. [5] generalized the work of Shapley and Shubik to matchings in general graphs as well as to cooperative games of many other combinatorial optimization problems. We refer the reader also to the recent survey paper [4] and the excellent textbook [2].

In this paper we further generalize the work of [10] and [1] on network bargaining by allowing contracts to span more than two agents. Our study is motivated by bargaining settings where goods are complex composites of other goods that are under the control of autonomous agents. For example, in a computer network setting, two hosts  $A$  and  $B$  may wish to establish a connection between themselves. Any such connection may involve physical links from a number of smaller autonomous networks that are provisioned by individual players. In this setting, value generated by the connection between  $A$  and  $B$  cannot merely be shared by the two hosts, but must also be used to compensate those *facilitators* whose networks the connection uses.

## 1.1 Generalized Network Bargaining

We formalize the above ideas by defining the class of *generalized network bargaining* (GNB) games. In an instance of such a game, we are given a (directed or undirected) graph  $G = (V, E)$  whose vertices correspond to players, and edges that correspond to atomic goods; the *value* of the good corresponding to  $e$  is given by  $w_e \geq 0$ . We assume that  $V$  is partitioned into *terminals*  $T$ , and *facilitators*  $R$ . Intuitively, the terminals are the *active* players that seek participation in contracts, while facilitators are *passive*, and may get involved in contracts, but do not seek involvement. We further let  $\mathcal{C}$  be a family of *contracts* each of whom consists of a collection of atomic goods. We let  $w(c)$  be the *value* of contract  $c$  which we simply define as the sum of values  $w_e$  of the edges  $e \in c$ . We note here that in the work of [10] and [1],  $\mathcal{C}$  consists just of the singleton edges.

A set  $C \subseteq \mathcal{C}$  of contracts is called *feasible* if each two contracts in  $C$  are vertex disjoint. An *outcome* of an instance of GNB is given by a feasible collection  $C \subseteq \mathcal{C}$  as well as an allocation  $x \in \mathbb{R}_+^V$  of the contract values to the players such that  $x(c) := \sum_{v \in c} x_v = w(c)$ .

Which outcomes are *desirable*? We propose the following natural extensions of the notions of *stability* and *balance* of [10]. Consider an outcome  $(C, x)$  of some instance of GNB. Then define the *outside option*  $\alpha_i$  of player  $i$  as  $\alpha_i := \max_{c \in \mathcal{C}: i \in c \notin C} \{w(c) - x(c)\} + x_i$ . Intuitively, the outside option of  $i$  is given by the value she can earn by breaking her current contract, and participating in a contract that is not part of the current outcome. We will assume that each agent  $i$  is incident to a self-loop of value 0, and hence has the option of not collaborating with anyone else. In what follows  $a(c) := \sum_{v \in c} a_v$  for a contract  $c \in \mathcal{C}$ .

Having defined  $\alpha_i$ , we can now introduce the notions of *stability* and *balance*. An outcome  $(C, x)$  is stable if  $x_i \geq \alpha_i$  for all agents  $i$ : every agent earns at least her outside option. Again extending the concept of Nash bargaining solution in the most natural way, we say that an outcome is balanced if the surplus of each contract is shared evenly among the participating agents. Formally, for all  $c \in C$ , and for all  $i \in c$  we require  $x_i = \alpha_i + \frac{w(c) - \alpha(c)}{|c|}$ . Equivalently, this means that  $x_i - \alpha_i = x_j - \alpha_j$  for all  $i, j \in c$ , and for all  $c \in C$ .

## 1.2 Our Results

Following Kleinberg and Tardos, we are interested in (a) characterizing the class of GNB instances that have stable and balanced outcomes, and (b) in computing such outcomes efficiently whenever they exist. Similar to [1], we first identify a natural *cooperative* game  $\Gamma(I)$  associated with a given GNB instance  $I$ .  $\Gamma(I)$  has player set  $V$  and the value function is defined by letting  $v(S) = \max_{\substack{C \subseteq \mathcal{C}(S), \\ C \text{ feasible}}} \sum_{c \in C} w(c)$ , for all  $S \subseteq V$ , where  $\mathcal{C}(S)$  is the set of contracts contained in the set  $S$ . We briefly introduce a few pertinent solution concepts for cooperative games, and refer the reader to [2] for a comprehensive introduction to the topic. The *core*  $\mathbf{C}$  of  $\Gamma(I)$  consists of all allocations  $x \in \mathbb{R}_+^V$  that satisfy  $x(S) \geq v(S)$  for all  $S \subseteq V$ , and this inequality is tight for  $S = V$ . The *power* of agent  $i$  over agent  $j$  is given by

$$s_{ij}(x) = \max_{S \subseteq V: i \in S, j \notin S} v(S) - x(S), \quad (1)$$

and the *prekernel*  $\mathbf{P}$  consists of all allocations  $x$  for which  $s_{ij} = s_{ji}$  for all agents  $i$  and  $j$ . In the following first result, let **stable** be the projection of the collection of all stable outcomes  $(C, x)$  onto the  $x$ -space. Similarly, **balance** is the projection of all balanced outcomes onto the  $x$ -space.

**Theorem 1.** *Let  $I$  be an instance of GNB, and  $\Gamma(I)$  the corresponding cooperative game. Then  $\mathbf{C} = \text{stable}$ , and  $\mathbf{C} \cap \mathbf{P} \subseteq \text{stable} \cap \text{balance}$ . There are instances of GNB where this inclusion is strict.*

It is well known (e.g., see [7]) that if the core of  $\Gamma(I)$  is non-empty then so is the intersection of core and prekernel. We therefore obtain the following corollary.

**Corollary 1.** *Every GNB instance with a stable solution also admits a balanced one.*

But can one find stable and balanced solutions efficiently? As it turns out (see below) not always. However, given a point in the core, and an efficient oracle for the computation of powers (as specified in (1)), we can find a point in the prekernel of  $\Gamma(I)$  via a result by Faigle, Kern and Kuipers [7] (see also [11]). We obtain the following corollary.

**Corollary 2.** *There is a polynomial-time algorithm to compute stable and balanced solutions for an instance of GNB if (a) we have a polynomial-time method to compute a point in the core, and (b) (1) can be computed efficiently.*

Unfortunately, computing  $s_{ij}$  in (1) may amount to solving an NP-hard problem; e.g., when  $\mathcal{C}$  consists of all paths in the given graph, one easily sees that a poly-time oracle for computing powers would enable us to solve the NP-hard *longest path* problem. Nevertheless, there are many families of instances of interest where the conditions of Corollary 1 are satisfied, e.g. instances where  $\mathcal{C}$  is explicitly given as part of the input, or whenever the family  $\mathcal{C}$  induces an acyclic subgraph of the input graph.

In light of Corollary 1, in order to characterize instances of GNB that have stable and balanced solutions, we may characterize the set of instances  $I$  for which  $\Gamma(I)$  has a non-empty core. We can show the following.

**Theorem 2.** *For a given GNB instance  $I$ , we can write a linear program  $(P_1)$  that has an integral optimal solution iff the core of  $\Gamma(I)$  is non-empty.*

Hence  $(P_1)$  fully characterizes the class of GNB instances that admit stable and balanced solutions. We may, however, not be able to solve the LP.

**Theorem 3.** *Given an instance  $I$  of GNB, it is NP-complete to (a) check whether the core of  $\Gamma(I)$  is non-empty, and (b) check whether a specific allocation  $x \in \mathbb{R}_+^n$  is in the core.*

In this theorem, we assume that  $\mathcal{C}$  is part of the input. We can show (a) by using a reduction from *exact-cover by 3-sets* following a previous result by Conitzer and Sandholm [3] closely. Part (b) employs a reduction from *3-dimensional matching*, and is similar to a result for minimum-cost spanning tree games by Faigle et al. [6].

We note here that the results in Theorems 1, 2 and 3 do not rely on the specific type of underlying graph (i.e., *directed* or *undirected*). Departing from this, our next

result focuses on GNB instances whose contract set is implicitly given as the set of all terminal-terminal paths in a *directed* graph. For such instances  $I$ , we present efficiently solvable linear programs ( $P_2$ ) and ( $P_3$ ) that are integral only if the core of  $\Gamma(I)$  is non-empty.

**Theorem 4.** *Given an instance  $I$  of GNB where  $\mathcal{C}$  is the set of all terminal-terminal paths in an underlying directed graph, we can find efficiently solvable LPs ( $P_2$ ) and ( $P_3$ ) that are integral only if the core of  $\Gamma(I)$  is non-empty.*

Unfortunately, the latter two LPs do not fully characterize core non-emptiness of  $\Gamma(I)$ , and there are instances with non-empty core for which the two LPs are fractional. The two LPs are not equivalent, and there are instances of GNB where one of the two LPs is fractional and the other is not.

## 2 Computing Balanced Outcomes

The goal of this section is to provide a proof of Theorem 1. Let us fix an instance  $I$  of GNB with graph  $G = (V, E)$ , and weights  $w_e$  for all  $e \in E$ . Recall that the cooperative game  $\Gamma(I)$  for  $I$  has player set  $V$ , and that the value  $v(S)$  of a coalition  $S \subseteq V$  is given by the maximum value of any feasible collection of contracts that are entirely contained in  $S$ . We first make the following observation.

**Observation 5.** *Computing  $v(V)$  for Distant Bargaining Games  $G=(V,E)$ , for which the set of feasible contracts is part of the input, is NP-hard.*

*Proof.* The reduction is from 3-dimensional matching (3DM) where all three vertex sets have the same size. Given an instance  $H = (L \cup M \cup R, F)$  of this problem (where  $F$  contains hyperedges, each containing exactly one vertex from each  $L, M, R$ ), we consider the following Distant Bargaining instance. The set of terminals is  $L \cup R$ , and the set of facilitators is  $M$ . For every hyperedge, we introduce two new edges connecting each of the two terminals to the facilitator, each of weight  $1/2$ , as well as we introduce the associated contract (of weight 1) containing exactly these two edges; we still allow only the “hyperedge contracts” to be formed by the new edges, and no other combinations. Finally, it is easy to see that  $H = (L \cup M \cup R, F)$  admits a 3d-matching if and only if  $v(L \cup M \cup R) = |L|$ .

We will now relate the core of  $\Gamma(I)$  and the set of stable outcomes of  $I$ . In order to do this, we need the following lemma, and leave its straight-forward proof to the reader.

**Lemma 1.** *Let  $x$  be an allocation in the core. Then there is a feasible collection  $C \subseteq \mathcal{C}$  of maximum value such that  $\sum_i x_i = \sum_{c \in C} w(c)$ .*

The following lemma, whose proof can be found in the full paper shows that core and set of stable allocations coincide.

**Lemma 2.**  *$C = \text{stable}$*

We now show that solutions in the prekernel  $\mathbf{P}$  are balanced.

**Lemma 3.**  $\mathbf{C} \cap \mathbf{P} \subseteq \text{stable} \cap \text{balance}$ 

*Proof.* Let  $x \in \mathbf{C} \cap \mathbf{P}$ . By Lemma 2 we know that  $x \in \text{stable}$ . Hence it remains to argue that  $x$  is also balanced. We first argue that for all agents  $i, j$ , whenever  $x \in \mathbf{C}$ , there must be a contract  $c$  containing  $i$  and not  $j$  such that  $s_{ij} = v(c) - x(c)$ . Indeed, suppose that  $s_{ij} = v(S) - x(S)$ , for some  $S \subseteq V$  for which  $i \in S \not\ni j$ . Then let  $c_1, \dots, c_t \in \mathcal{C}(S)$  be a feasible collection of contracts whose joint value equals  $v(S)$ . Without loss of generality, suppose that  $i \in c_1$ . Then for every contract  $c_r$ ,  $x \in \mathbf{C}$  implies that  $w(c_r) - x(c_r) \leq 0$ , and hence, as claimed,

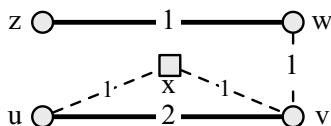
$$v(S) - x(S) = \sum_{r=1}^t (w(c_r) - x(c_r)) \leq w(c_1) - x(c_1).$$

Since  $x \in \mathbf{P}$  we know that  $s_{ij}(x) = s_{ji}(x)$ , for all  $i, j \in V$ . From Lemma 1 we also know that  $x$  corresponds to some maximum value set of feasible contracts, say,  $C$ . Fix a contract  $c \in C$  and two agents  $i, j \in C$ . In what follows we argue that  $\alpha_i - x_i = \alpha_j - x_j$ , which directly implies that  $x \in \text{balance}$ .

For the sake of simplicity, we denote  $\arg \max \alpha_i$  and  $\arg \max s_{ij}(x)$  by  $c_i$  and  $q_{ji}$  respectively. Then we note that if  $i \notin c_j$  then  $q_{ji} = c_j$  and hence  $s_{ji}(x) = \alpha_j - x_j$ . Also, if  $i \in c_j$ , then  $\alpha_j - x_j \geq s_{ji}(x)$ , since the set  $c_j$  is not considered when we maximize over subsets in order to find  $s_{ij}(x)$ .

With these observations at hand, we can now examine three cases. First, suppose that  $i \notin c_j$  and  $j \notin c_i$ . Then  $s_{ij}(x) = s_{ji}(x)$  implies that  $\alpha_i - x_i = \alpha_j - x_j$ . In the second case, if  $i \in c_j$  and  $j \in c_i$ , then  $c_i = c_j$ , so again  $\alpha_i - x_i = \alpha_j - x_j$ . Finally, in the third case we assume that  $i \notin c_j$  (and therefore  $q_{ji} = c_j$ , that is  $s_{ji}(x) = \alpha_j - x_j$ ) and that  $j \in c_i$  (and so,  $\alpha_i - x_i \geq s_{ij}(x)$ ). It follows that if  $s_{ij}(x) = s_{ji}(x)$ , then  $\alpha_i - x_i \geq \alpha_j - x_j$ . Also, since  $j \in c_i$  we conclude that  $\alpha_j - x_j \geq \alpha_i - x_i$ . Overall, this implies again that  $\alpha_j - x_j = \alpha_i - x_i$ , as we wanted.

Figure 1 shows an instance of GNB with terminals  $\{u, v, w, z\}$  and contracts  $\mathcal{C} = \{uv, vw, wz, uz, uxv\}$ . Consider feasible contracts  $C = \{uv, wz\}$  of total value 3. The allocation  $\chi_u = \chi_v = 1$ ,  $\chi_w = \chi_z = 1/2$ , and  $\chi_x = 0$  is easily checked to be stable and balanced. However, since  $s_{uv} = 0 - (1/2 + 1) = -3/2$ , and  $s_{vu} = 1 - (1/2 + 1) = -1/2$ ,  $x$  is not in the prekernel. Together with Lemmata 2 and 3 we obtain a proof of Theorem 1.



**Fig. 1.** Counter-example for the reverse inclusion in Lemma 3

### 3 Characterizing the Core

As we have seen in Lemma 2, the set of stable allocations for a GNB instance  $I$  equals the core of the cooperative game  $\Gamma(I)$ . In this section, our goal will be to characterize instances  $I$  where  $\Gamma(I)$  has a non-empty core. Further, if the core of  $\Gamma(I)$  is non-empty then we will investigate the computational complexity of finding such a point.

#### 3.1 The Core via Linear Programming

We start this section by presenting a linear programming formulation that is integral iff the core of  $\Gamma(I)$  is non-empty. The LP has a variable  $z_c$  for each contract  $c \in \mathcal{C}$ , and maximizes the total value of chosen contracts subject to feasibility. The LP is shown on the left below.

$$\begin{array}{ll} \max & \sum_{c \in \mathcal{C}} w(c) z_c \\ \text{s.t.} & \sum_{c: i \in c} z_c \leq 1, \quad \forall i \in V \\ & z \geq 0 \end{array} \quad (\mathbf{P}_1) \quad \left| \quad \begin{array}{ll} \min & \sum_{i \in V} y_i \\ \text{s.t.} & \sum_{i \in c} y_i \geq w(c), \quad \forall c \in \mathcal{C} \\ & y \geq 0. \end{array} \right. \quad (\mathbf{D}_1)$$

The LP on the right is the linear programming dual of  $(\mathbf{P}_1)$ . It has a variable  $y_i$  for each agent  $i \in V$ , and a constraint for every  $c \in \mathcal{C}$ . We will now present a proof of Theorem 2, and show that the core of  $\Gamma(I)$  is non-empty iff  $(\mathbf{P}_1)$  has an integral optimal solution.

*Proof (Proof of Theorem 2).* Recall that by Lemma 2, the core of  $\Gamma(I)$  equals the set **stable** of stable allocations in  $I$ . Also recall that an outcome  $(C, x)$  is stable iff for all  $c' \notin C$ ,  $x(c') \geq w(c')$  and  $w(c) = x(c)$  for all  $c \in C$ .

Now suppose that  $(\mathbf{P}_1)$  has an integral optimal solution  $z$ , and let  $y$  be the corresponding optimal dual solution. Clearly,  $0 \leq z \leq \mathbb{1}$ , and hence we may define the set  $C \subseteq \mathcal{C}$  of contracts  $c$  with  $z_c = 1$ . We now claim that  $(C, y)$  is a stable outcome. Indeed, all stability conditions are provided by the dual constraints, and by complementary slackness, they are tight when  $z_c > 0$ .

For the other direction, consider a stable outcome  $(C, x)$ . It is easy to see that  $z_P = 1$  for  $P \in C$  and 0 otherwise, and  $y = x$  are primal and dual feasible solutions respectively. Complementary slackness is implied exactly by the definition of outcomes that require that the sum of agent earnings in each contract matches the contract surplus.

We do not know how to solve  $(\mathbf{P}_1)$  efficiently. Worse than that, even if we are able to solve the LP, we may not be able to decide whether there is an integral optimal solution. The proof of the following result is implicit in [3], and given here for completeness.

**Lemma 4.** *Given an instance  $I$  of GNB it is NP-complete to decide whether the core of  $\Gamma(I)$  is non-empty.*

*Proof.* We first show that the problem is in NP. For this, we non-deterministically guess a feasible collection  $C \subseteq \mathcal{C}$  of contracts. We then solve the linear system

$$x(c) = w(c) \quad \forall c \in C \quad (2)$$

$$x(c) \geq w(c) \quad \forall c \in \mathcal{C} \setminus C. \quad (3)$$

in order to find  $x \in \mathbb{R}_+^V$ . This can be done in polynomial time (e.g., via linear programming) as  $\mathcal{C}$  is part of the input. It is easy to check that the system has a feasible solution if  $x$  is in the core.

To show hardness, we reduce from an instance of *exact cover by 3-sets* (X3C), where we are given a ground-set  $S$  of size  $3m$  and subsets  $\{S_1, \dots, S_q\}$  of  $S$  each of which has size 3. The question is whether there are  $m$  pairs whose union is  $S$ .

Here is how we encode this problem as an instance of GNB. We create a graph  $G$  with vertex set  $S \cup \{x, y\}$ , where  $x$  and  $y$  are two new dummy vertices. For each  $S_i = \{a, b, c\}$  in the X3C instance, we add distinct edges  $ab$  and  $bc$  each of cost  $3/2$  (middle vertex is chosen, say, lexicographically), and we add  $abc$  to the list of allowed contracts  $\mathcal{C}$ . We also add trees  $T_x$  and  $T_y$  spanning  $S \cup \{x\}$ , and  $S \cup \{y\}$ , respectively. Once again, the edge sets of  $T_x$  and  $T_y$  are disjoint, and distinct from the other edges added previously. We distribute weight  $6m$  over the edges of  $T_x$ , and similarly over the edges of  $T_y$  in some arbitrary way, and add  $E(T_x)$  and  $E(T_y)$  to the set of allowed contracts. Finally we add  $xy$  to the graph and contract set, and assign a weight of  $6m$  to this edge. We claim that the core of the game  $\Gamma(I)$  of the above instance is non-empty iff the given X3C instance is a 'yes' instance.

Assume first that  $S_1, \dots, S_m$  is an exact 3-cover. In this case, note that the corresponding contracts together with  $xy$  are feasible, and have joint value  $9m$ . One can now verify that  $\chi$  with  $\chi_i = 1$ , for all  $i \in S$ , and  $\chi_x = \chi_y = 3m$  is in the core.

Conversely, if no exact 3-cover exists, then the value  $v(S)$  is less than  $3m$ , and the value of the grand coalition is less than  $9m$ . Consider any vector  $\chi \in \mathbb{R}_+^{S \cup \{x, y\}}$  such that  $\chi(V) = v(V) < 9m$ . It is not difficult to see that there are two distinct sets  $U, W \in \{S, \{x\}, \{y\}\}$ , such that  $\chi(U) + \chi(W) < 6m$ . But  $U \cup W$  is a coalition of value  $6m$  by our definition, and  $\chi$  is therefore not in the core.

So, even if (P<sub>1</sub>) can be solved efficiently, we may not be able to check efficiently for an integral optimal solution. We now show that it is also hard to check whether a certain allocation is in the core, which in combination with Lemma 4 conclude Theorem 3.

**Lemma 5.** *It is NP-complete to check whether an allocation  $x \in \mathbb{R}_+^V$  is in the core of the cooperative game of a GNB instance  $I$ .*

*Proof.* The problem is certainly contained in NP. To see this, we first nondeterministically guess a feasible collection  $C \subseteq \mathcal{C}$  and then check that (2) and (3) hold.

To prove hardness, we once again reduce from the 3-dimensional matching problem. Given an instance of 3DM, we create an instance of GNB by creating a graph with terminal vertices  $L \cup M \cup R$ . For each  $(l, m, r) \in F$ , we add edges  $lm$  and  $mr$  of value  $1/2$  each, and add contract  $\{lm, mr\}$  to the set  $\mathcal{C}$  of allowed contracts.

Consider the vector  $\chi$  with  $\chi_v = 1$  if  $v \in M$ , and  $\chi_v = 0$  otherwise. We claim that  $\chi$  is in the core iff the given 3DM instance is a 'yes' instance.

If the given 3DM instance is a 'no' instance, then  $v(V) < |M| = \chi(V)$ , and hence  $\chi$  is not in the core. Conversely suppose that the 3DM instance is a 'yes' instance. In this case,  $\chi(V) = |M| = v(V)$ , and clearly  $\chi(c) = 1 = w(c)$  for every contract  $c \in C$ .

### 3.2 Implicitly Given Contracts

In this section, we focus on GNB instances where  $\mathcal{C}$  is implicitly given as the set of *all terminal-terminal paths* in an underlying directed graph  $D$  with node-set  $V$ , and arcs  $A$ . The internal nodes of each of these paths are assumed to be facilitators. Thus,  $\mathcal{C}$  is not part of the input, and LP (P<sub>1</sub>) may have an exponential number of variables. We do not know how to efficiently solve this LP in this case. In the following, we present two LPs for a given instance of GNB that (a) have integral optimal solutions only if the core of  $\Gamma(I)$  is non-empty, and (b) are poly-time solvable.

In the following, let us fix an instance  $I$  of GNB with graph  $D = (V, A)$ , and weights  $w_{uv}$  for all arcs  $(u, v) \in A$ . The two LPs to be presented are *flow* formulations.

**Cycle-Free Flow Formulation.** Observe that a set  $C$  of arcs in  $D$  corresponds to a feasible set of contracts iff (a) every terminal agent has at most one incident arc, (b) every facilitator agent has at most one outgoing arc, (c) every facilitator has equally many incoming and outgoing arcs, and (d) for every set of facilitators  $S$ , there is at least one outgoing arc in  $C$  if there is an arc in  $C$  that has both endpoints in  $S$ . Therefore, the following LP is a relaxation for computing the value of the grand coalition (recall that contracts are all terminal-terminal paths). For a set  $S$  of nodes, we let  $\delta^+(S)$  be the set of arcs with tail in  $S$  and head outside  $S$ . Furthermore, we let  $\gamma(S)$  be the set of arcs with both ends in  $S$ .

$$\begin{aligned} \max \quad & \sum_{a \in A} w_a x_a && (\text{P}_2) \\ \text{s.t.} \quad & x(\delta^-(v)) + x(\delta^+(v)) \leq 1 && \forall v \in T \\ & x(\delta^+(v)) \leq 1 && \forall v \in R \\ & x(\delta^-(v)) - x(\delta^+(v)) = 0 && \forall v \in R \\ & x(\delta^+(S)) \geq x_a && \forall S \subseteq R, \forall a \in \gamma(S) \\ & x \geq 0 && \end{aligned} \quad (4)$$

Note that (P<sub>2</sub>) can be solved in polynomial time via the Ellipsoid method [9]: given a candidate solution  $x$ , it can be efficiently checked whether one of the polynomially many constraints of one of the first three types is violated. Separating the constraints of type (4) can be reduced to a polynomial number of minimum-cut computations in suitable auxiliary graphs. We leave the details to the reader.

**Lemma 6.** *If LP (P<sub>2</sub>) for GNB instance  $I$  has an integral optimal solution, then the core of  $\Gamma(I)$  is non-empty.*

The reader can find a constructive proof of Lemma 6 in the full version of the paper. Note that Lemma 6 is a direct implication of Theorem 2. The reason is that any solution feasible to (P<sub>1</sub>) can be converted to a feasible solution to (P<sub>2</sub>) (of equal objective value) as follows: for every  $uv \in A$  set  $\chi_{uv} = \frac{1}{2} \sum_{P \in \mathcal{P}: uv \in P} z_P$ . It follows that the optimal value of (P<sub>1</sub>) is sandwiched between the value of the cooperative game  $\Gamma(I)$ , and that

of  $(P_2)$ . Taking into consideration that  $(P_2)$  restricted to integral values is an exact formulation of our problem, if  $(P_2)$  has integrality gap 1, so does  $(P_1)$ . Nevertheless, the important observation is that unlike  $(P_1)$ , we know how to efficiently solve relaxation  $(P_2)$ . Furthermore, the proof of Lemma 6 is constructive, and gives rise to an efficient algorithm to compute a core allocation.

Unfortunately, we will later see that there are example instances of GNB with non-empty core for which  $(P_2)$  has no integral optimal solution (see Lemma 8). There are, however, many natural instance classes of GNB for which we are able to find core allocations if these exist via our LP. An example is a class of multi-layered graphs where the nodes are partitioned in  $k$  layers  $L_1, \dots, L_k$ , with  $(L_1 \cup L_2) = T$ ,  $(L_2 \cup L_3 \cup \dots \cup L_{k-1}) = R$ , and arcs existing only between nodes in consecutive layers (i.e., if  $(u, v) \in E$  then  $u \in L_i, v \in L_{i+1}$  for some  $1 \leq i \leq k-1$ ). Note that the only contracts allowed are paths from terminals in  $L_1$  to terminals in  $L_k$ . Each feasible solution we get if we relax  $(P_2)$  by removing constraints (4) can be mapped to a single-commodity flow on the network we get if we connect all nodes in  $L_1$  to a source  $s$  and all nodes in  $L_k$  to a sink  $t$  (with arcs of weight 0), and we give capacity 1 to all nodes. Each such flow can also be mapped to a feasible solution of the relaxed  $(P_2)$ . Since the flow polytope is integral, the optimal solution of the relaxed  $(P_2)$  is also integral; it also satisfies constraints (4), and, therefore, it is also an integral optimal solution for  $(P_2)$ .

**Subtour Formulation.** We now present yet another polynomial-time solvable relaxation for GNB. Once again, we will see that the existence of an integral optimal solution implies non-emptiness of core for  $\Gamma(I)$ , and that the reverse of this statement is false. However, we show in the full version of the paper that  $(P_2)$  and this new LP are incomparable, and one may have integer optimal solution when the other does not.

$$\begin{aligned} \max \quad & \sum_{a \in A} w_a x_a \\ \text{s.t.} \quad & x(\delta^+(v)) + x(\delta^-(v)) \leq 1 \quad \forall v \in T \\ & x(\delta^+(v)) \leq 1 \quad \forall v \in R \\ & x(\delta^-(v)) - x(\delta^+(v)) = 0 \quad \forall v \in R \\ & x(\gamma(S)) \leq |S| - 1 \quad \forall S \subseteq R \\ & x \geq 0 \end{aligned} \tag{P}_3$$

It can be easily seen that  $(P_3)$  restricted on integral values models exactly the problem of computing the value of the grand coalition in the associated coalition game (recall that contracts are all terminal-terminal paths). Once again it is easily shown that  $(P_3)$  is polynomial-time solvable, and once again we utilize the Ellipsoid method. We observe that the function  $(|S| - 1) - x(\gamma(S))$  is submodular. Separating the constraints of type (4) then reduces to submodular function minimization, for which there are polynomial-time algorithms (e.g., see [13]).

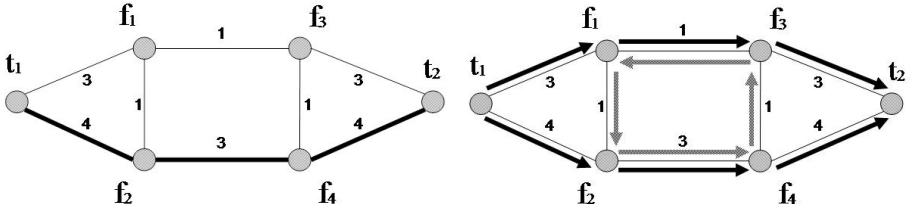
Similarly to the previous section, we show that  $(P_3)$  can be used as a certificate that the core is non empty, for some, but not all instances.

**Lemma 7.** *If LP  $(P_3)$  for GNB instance  $I$  has an integral optimal solution, then the core of  $\Gamma(I)$  is non-empty.*

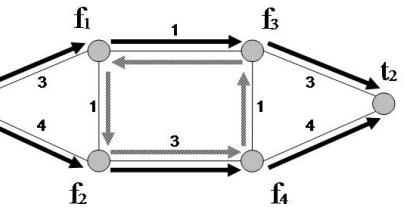
Similarly to  $(P_2)$ , Lemma 7 is a direct consequence of Theorem 2. We provide a constructive proof in the full version of the paper which gives rise to an efficient algorithm for computing core allocations. Finally note that Lemmata 6 and 7 prove Theorem 4.

**Lemma 8.** *There are instances for which the  $C$  is non empty, still the integrality gap of both  $(P_2)$  and  $(P_3)$  is bigger than 1.*

*Proof.* Consider terminals  $t_1, t_2$  and facilitators  $f_1, f_2, f_3, f_4$ , with edges connecting them (along with weights) as seen in Figure 2. One of the optimal solutions is the



**Fig. 2.** The optimal contract



**Fig. 3.** The fractional LP solution

path  $t_1f_2, f_2f_4, f_4t_2$  of value 11. A core assignment would give  $x_{t_1} = x_{t_2} = \frac{11}{2}$ , and 0 to all facilitators. This can be seen to be in the core since contracts are always paths connecting  $t_1, t_2$ , and none of them has cost more than what both terminals earn together.

Finally, we argue how to fool both  $(P_2)$  and  $(P_3)$ . For this we invent three flows; the path  $t_1f_2, f_2f_4, f_4t_2$ , the path  $t_1f_1, f_1f_3, f_3t_2$  and the cycle  $f_1f_2, f_2f_4, f_4f_3, f_3f_1$  (depicted in Figure 3) all with value 1/2. A claim that can be easily checked is that the proposed values satisfy both LPs, while the objective value in both cases is 12, which is strictly bigger than the integral optimal.

## 4 Conclusion

In this paper, we introduce the class of *generalized bargaining* games as a natural extension of *network bargaining*. We show that many of the known results for network bargaining extend to the new setting. For example, we show that an instance  $I$  of GNB has a balanced outcome whenever it has a stable one. We define a cooperative game  $\Gamma(I)$  for every GNB instance  $I$  and present an LP  $(P_1)$  that has an integral optimal solution iff the core of  $\Gamma(I)$  is non-empty.

Several interesting open questions remain: (1) In the case where the set of contracts is implicitly given as all terminal-terminal paths in the underlying graph, is it hard to solve  $(P_1)$  efficiently? (2) In the same setting, can we give a good characterization of the class of graphs (possibly via excluded minors) that have stable solutions?

## References

1. Bateni, M., Hajiaghayi, M., Immorlica, N., Mahini, H.: The cooperative game theory foundations of network bargaining games. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 67–78. Springer, Heidelberg (2010)
2. Chalkiadakis, G., Elkind, E., Wooldridge, M.: Computational Aspects of Cooperative Game Theory. Morgan & Claypool Publishers (2011)
3. Conitzer, V., Sandholm, T.: Complexity of constructing solutions in the core based on synergies among coalitions. *Artif. Intell.* 170(6-7), 607–619 (2006)
4. Deng, X., Fang, Q.: Algorithmic cooperative game theory. In: Pareto Optimality, Game Theory and Equilibria. Springer Optimization and Its Applications, vol. 17, pp. 159–185. Springer (2008)
5. Deng, X., Ibaraki, T., Nagamochi, H.: Algorithmic aspects of the core of combinatorial optimization games. *Math. Oper. Res.* 24(3), 751–766 (1999)
6. Faigle, U., Kern, W., Fekete, S.P., Hochstättler, W.: On the complexity of testing membership in the core of min-cost spanning tree games. *Int. J. Game Theory* 26(3), 361–366 (1997)
7. Faigle, U., Kern, W., Kuipers, J.: An efficient algorithm for nucleolus and prekernel computation in some classes of tu-games. Technical Report 1464, U. of Twente (1998)
8. Granot, D., Granot, F.: On some network flow games. *Math. Oper. Res.* 17(4), 792–841 (1992)
9. Grötschel, M., Lovász, L., Schrijver, A.: The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica* 1, 169–197 (1981)
10. Kleinberg, J.M., Tardos, É.: Balanced outcomes in social exchange networks. In: Proceedings of the ACM Symposium on Theory of Computing, pp. 295–304 (2008)
11. Meinhardt, H.: An lp approach to compute the pre-kernel for cooperative games. *Computers & OR* 33, 535–557 (2006)
12. Nash, J.: The bargaining problem. *Econometrica* 18, 155–162 (1950)
13. Schrijver, A.: Combinatorial optimization. Springer, New York (2003)
14. Shapley, L.S., Shubik, M.: The assignment game: the core. *International Journal of Game Theory* 1(1), 111–130 (1971)

# The 1/4-Core of the Uniform Bin Packing Game Is Nonempty

Walter Kern and Xian Qiu

Department of Applied Mathematics, University of Twente  
`kern@math.utwente.nl, x.qiu@utwente.nl`

**Abstract.** A cooperative bin packing game is an  $N$ -person game, where the player set  $N$  consists of  $k$  bins of capacity 1 each and  $n$  items of sizes  $a_1, \dots, a_n$ . The value of a coalition of players is defined to be the maximum total size of items in the coalition that can be packed into the bins of the coalition. We adopt the taxation model proposed by Faigle and Kern (1993) [6] and show that the 1/4-core is nonempty for all instances of the bin packing game. This strengthens the main result in [3].

## 1 Introduction

Since many years, logistics and supply chain management are playing an important role in both industry and our daily life. In view of the big profit generated in this area, the question therefore arises how to “fairly” allocate profits among the “players” that are involved. Take online shopping as an example: Goods are delivered by means of carriers. Generally, shipping costs are proportional to the weight or volume of the goods, and the total cost is basically determined by the competitors. But there might be more subtle ways to compute “fair” shipping cost (and allocation between senders and receivers). It is natural to study such allocation problems in the framework of cooperative games. As a first step in this direction we analyze a simplified model with uniform packet sizes as described in more detail in section 2.

A *cooperative game* is defined by a tuple  $\langle N, v \rangle$ , where  $N$  is a set of *players* and  $v : 2^N \rightarrow \mathbb{R}$  is a *value function* satisfying  $v(\emptyset) = 0$ . A subset  $S \subseteq N$  is called a *coalition* and  $N$  is called the *grand coalition*. The usual goal in cooperative games is to “fairly” allocate the total gain  $v(N)$  of the grand coalition  $N$  among the individual players. A well known concept is the *core* of a cooperative game, defined by all *allocation vectors*  $x \in \mathbb{R}^N$  satisfying

- (i)  $x(N) = v(N)$ ,
- (ii)  $x(S) \geq v(S)$  for all  $S \subseteq N$ .

As usual, we abbreviate  $x(S) = \sum_{i \in S} x_i$ .

We say a cooperative game is balanced if there exists a core allocation for any instance. Unfortunately, many games are not balanced. Players in a non-balanced game may not cooperate because no matter how the total gain is distributed,

there will always be some coalition  $S$  with  $x(S) < v(S)$ , i.e., it gets paid less than it could earn on its own. For this case, one naturally seeks to relax the condition (ii) above in such a way that the modified core becomes nonempty. Faigle and Kern [6] introduced the multiplicative  $\epsilon$ -core as follows. Given  $\epsilon > 0$ , the  $\epsilon$ -core consists of all vectors  $x \in \mathbb{R}^N$  satisfying condition (i) above together with

$$(ii') \quad x(S) \geq (1 - \epsilon)v(S) \text{ for all subsets } S \subseteq N.$$

We can interpret  $\epsilon$  as a tax rate in the sense that coalition  $S$  is allowed to keep only  $(1 - \epsilon)v(S)$  on its own. Thus, the  $\epsilon$ -core provides an  $\epsilon$ -approximation to balancedness. If the value function  $v$  is nonnegative, the 1-core is obviously nonempty. In order to approximate the core as close as possible, one would like to have the taxation rate  $\epsilon$  as small as possible while keeping the  $\epsilon$ -core nonempty. Discussions of the  $\epsilon$ -core allocation for other cooperative games, say, facility location games, TSP games etc. can be found in [10],[11],[8],[5].

As motivated at the beginning of this paper, we study a specific game of the following kind: There are two disjoint sets of players, say,  $A$  and  $B$ . Each player  $i \in A$  possesses an item of value/size  $a_i$ , for  $i = 1, \dots, n$ , and each player  $j \in B$  possesses a truck /bin of capacity 1. The items produce a profit proportional to their size  $a_i$  if they are brought to the market place. The value  $v(N)$  of the grand coalition thus represents the maximum profit achievable. How should  $v(N)$  be allocated to the owners of the items and the owners of the trucks?

Bin packing games were first investigated by Faigle and Kern [6]. In their paper, they considered the non-uniform bin packing game (i.e., bin capacities are distinct) and showed that the  $1/2$ -core is nonempty if any item can be packed to any bin. Later, researchers focused on the uniform case, where all bins have the same capacity. Woeginger [17] showed that the  $1/3$ -core is always nonempty. Recently, Kern and Qiu [13] improved this result to  $1/3 - 1/108$ . Eventually, however, it turned out that the standard (unmodified) version of the set packing heuristic, combined with the matching arguments already used in [7] and yet another heuristic based on replacing complete bins as described in detail in section 3 led to the improvement  $\epsilon_N \geq 1/4$ . As  $v'(N) \geq v(N)$ , this amounts to a strengthening of the main result in [3], hopefully also providing additional insight.

Other previous results on bin packing games can be summarized as follows. In [7], Faigle and Kern show that the integrality gap, defined by the difference of the optimum fractional packing value and the optimum integral packing value (*cf.* section 2) is bounded by  $1/4$ , if all item sizes are strictly larger than  $1/3$ , thereby implying that the  $1/7$ -core is nonempty in that case (which was independently shown by Kuipers [15]). Moreover, in the general case, given a fixed  $\epsilon \in (0, 1)$ , they prove that the  $\epsilon$ -core is always non-empty if the number of bins is sufficiently large ( $k \geq O(\epsilon^{-5})$ ). Liu [16] gives complexity results on testing emptiness of the core and core membership for bin packing games, stating that both problems are NP-complete. Also, Kern and Qiu [14] studied the non-uniform bin packing game and showed that the  $1/2$ -core is nonempty for any instance. Moreover, the problem of approximating the maximum packing value  $v(N)$  is also studied in literature (called “multiple subset sum problem”): PTAS and  $3/4$  approximation

algorithms are proposed in [2] and [3]. Other variants of the bin packing game can also be found in [1], [18], [4], [9] etc..

The rest of the paper is organized as follows. In section 2, we give a formal definition of bin packing games and introduce fractional packings. In section 3, we prove that the 1/4-core is always nonempty. Finally, in section 4, we remark on some open problems.

## 2 Preliminaries

A *bin packing game*  $N$  is defined by a set  $A$  of  $n$  items  $1, 2, \dots, n$  of sizes  $a_1, a_2, \dots, a_n$ , and a set  $B$  of  $k$  bins, of capacity 1 each, where we assume, w.l.o.g.,  $0 \leq a_i \leq 1$ .

A *feasible packing* of an item set  $A' \subseteq A$  into a set of bins  $B' \subseteq B$  is an assignment of some (or all) elements in  $A'$  to the bins in  $B'$  such that the total size of items assigned to any bin does not exceed its capacity. Items that are assigned to a bin are called *packed* and items that are not assigned are called *not packed*. The *value* or *size* of a feasible packing is the total size of packed items.

The player set  $N$  consists of all items and all bins. The value  $v(S)$  of a coalition  $S \subseteq N$ , where  $S = A_S \cup B_S$  with  $A_S \subseteq A$  and  $B_S \subseteq B$ , is the maximum value of all feasible packings of  $A_S$  into  $B_S$ . A corresponding feasible packing is called an *optimum packing*.

Let  $F$  be an item set, and denote by  $a_F = \sum_{i \in F} a_i$  the *value* of  $F$ .  $F$  is called a *feasible set* if  $a_F \leq 1$ . Denote by  $\mathcal{F}$  the set of all feasible sets, w.r.t. all items of  $N$ . Then the total earning  $v(N)$  of the grand coalition  $N$  equals

$$\begin{aligned} & \max \sum_{F \in \mathcal{F}} a_F y_F \\ & \text{s.t. } \sum_{F \in \mathcal{F}} y_F \leq k, \\ & \quad \sum_{\substack{F \ni i}} y_F \leq 1 \quad (i = 1, 2, \dots, n), \\ & \quad y_F \in \{0, 1\}. \end{aligned} \tag{2.1}$$

The value  $v'(N)$  of an *optimum fractional packing* is defined by the relaxation of (2.1), i.e.,

$$\begin{aligned} & \max \sum_{F \in \mathcal{F}} a_F y_F \\ & \text{s.t. } \sum_{F \in \mathcal{F}} y_F \leq k, \\ & \quad \sum_{\substack{F \ni i}} y_F \leq 1 \quad (i = 1, 2, \dots, n), \\ & \quad y_F \in [0, 1]. \end{aligned} \tag{2.2}$$

A *fractional packing* is a vector  $y$  satisfying the constraints of the linear program (2.2). We call a feasible set  $F$  *selected/packed* by a packing  $y'$  if  $y'_F > 0$ . Accordingly, we refer to the original “feasible packing” as *integral packing*, which meets the constraints of (2.1). To illustrate the definition of the fractional packing, we consider an example as follows.

*Example.* Given 2 bins and 4 items of sizes  $a_i = 1/2$  for  $i = 1, 2, 3$  and  $a_4 = 1/2 + \epsilon$ , with a small  $\epsilon > 0$ .

Obviously, packing items 1, 2 into the first bin and packing item 4 to the second bin results in an optimum integral packing of total value  $v(N) = 3/2 + \epsilon$ . Let  $F_1 = \{1, 2\}$ ,  $F_2 = \{2, 3\}$ ,  $F_3 = \{1, 3\}$ ,  $F_4 = \{4\}$ . By solving the linear program (2.2), the optimum fractional packing  $y' = (y'_F)$  selects  $F_1, \dots, F_4$  with a fraction  $1/2$  each, resulting in a value

$$v'(N) = \sum_{j=1}^4 y'_{F_j} a_{F_j} = \frac{7}{4} + \frac{\epsilon}{2} > v(N).$$

An intriguing problem is to find the “minimal” taxation rate  $\epsilon_{\min}$  such that the  $\epsilon_{\min}$ -core is nonempty for all instances of the bin packing game. Due to the following observation of Faigle and Kern in [7], this amounts to bounding the *relative gap*  $(v'(N) - v(N))/v'(N)$  for any bin packing game instance  $N$ .

**Lemma 1 ([7]).** *Given  $\epsilon \in (0, 1)$  and a bin packing game  $N$ , the  $\epsilon$ -core  $\neq \emptyset$  if and only if  $\epsilon \geq 1 - v(N)/v'(N)$ .*

We let  $\epsilon_N$  denote the relative gap of  $N$ . Trivially, if all items are packed in a feasible integral packing, we get  $v(N) = v'(N)$ , implying that  $\epsilon_N = 0$ , so the core is nonempty. Thus let us assume in what follows that no feasible integral packing packs all items. Clearly, any feasible integral packing  $y$  with corresponding packed sets  $F_1, \dots, F_k$  yields a lower bound  $v(N) \geq w(y) = \sum_{i=1}^k a_{F_i}$ . In view of Lemma 1 we are particularly interested in integral packings  $y$  of value  $w(y) \geq (1 - \epsilon)v'(N)$  for certain  $\epsilon > 0$ . For  $\epsilon = 1/2$ , such integral packings are easy to find by means of a simple *greedy packing* heuristic, that constructs a feasible set  $F_j$  to be packed into bin  $j$  in the following way: First order the available (yet unpacked) items non-increasingly, say,  $a_1 \geq a_2 \geq \dots$ . Then, starting with  $F_j = \emptyset$ , keep adding items from the list as long as possible (i.e.,  $a_{F_j} + a_i \leq 1 \Rightarrow F_j \leftarrow F_j \cup \{i\}$ , else proceed to  $i + 1$ ). Clearly, this eventually yields a feasible  $F_j$  of size  $> \frac{1}{2}$ . Indeed - unless all items get packed - the final  $F_j$  has size  $> 1 - a$ , where  $a$  is the minimum size of an unpacked item. Applying greedy packing to all bins will exhibit an integral packing  $y$  with  $a_{F_j} > \frac{1}{2}$  for all  $j$ , so  $w(y) > k/2 \geq v'(N)/2$ , thus proving non-emptiness of the  $\frac{1}{2}$ -core by Lemma 1.

A bit more work is required to exhibit integral packings with  $w(y) \geq \frac{2}{3}v'(N)$  (c.f. [17], [13]). We end this section by mentioning two trivial cases, namely  $k = 1$  and  $k = 2$ : Indeed, in case  $k = 1$ , we obviously have  $v(N) = v'(N)$ , thus  $\epsilon_N = 0$  and in case  $k = 2$ , the bound  $\epsilon_N \leq \frac{1}{4}$  can be seen as follows: Let  $y'$  be an optimal fractional solution and let  $F$  be a largest (most valuable) feasible set in the support of  $y'$ . Then the integral packing  $y$  that packs  $F$  into the first bin

and fills the second bin greedily to at least  $1/2$  (as explained above) is easily seen to yield a value  $w(y) \geq \frac{3}{4}w(y')$ , so that  $\epsilon_N \leq \frac{1}{4}$ .

### 3 Proof of Non-emptiness of the 1/4-Core

Throughout this section, we assume that  $N$  is a smallest counterexample, i.e., a game with  $1/4\text{-core}(N) = \emptyset$  and  $n + k$ , the number of players is as small as possible. We start with a simplification, similar to that in [17] for the case  $\epsilon = 1/3$ . The following result is a special case of Lemma 2.3 in [13], but we include a proof for convenience:

**Lemma 2.** *All items have size  $a_i > 1/4$ .*

*Proof.* Assume to the contrary that some item, say, item  $n$  has size  $a_n \leq 1/4$ . Let  $\bar{N}$  denote the game obtained from  $N$  by removing this item. Thus  $1/4\text{-core}(\bar{N}) \neq \emptyset$  or, equivalently,  $v(\bar{N}) \geq \frac{3}{4}v'(\bar{N})$ . Adding item  $n$  to  $\bar{N}$  can clearly not increase  $v'$  by more than  $a_n$ , i.e.,

$$v'(N) \leq v'(\bar{N}) + a_n. \quad (3.1)$$

Let  $\bar{y}$  be any optimal integer packing for  $\bar{N}$ , i.e.,  $w(\bar{y}) = v(\bar{N})$ . Then either item  $n$  can be packed “on top of  $\bar{y}$ ” (namely when some bin is filled up to  $\leq 1 - a_n$ ) – or not. In the latter case, the packing  $y$  fills each bin to more than  $1 - a_n \geq 3/4$ , thus  $w(\bar{y}) \geq \frac{3}{4}k \geq \frac{3}{4}v'(N)$ , contradicting the assumption that  $\frac{1}{4}\text{-core}(N) = \emptyset$ . Hence, item  $n$  can indeed be packed on top of  $\bar{y}$ , yielding  $v(N) \geq w(\bar{y}) + a_n = v(\bar{N}) + a_n$ . Together with (3.1) this yields  $v'(N) - v(N) \leq v'(\bar{N}) - v(\bar{N})$  and hence

$$\epsilon_N = \frac{v'(N) - v(N)}{v'(N)} \leq \frac{v'(\bar{N}) - v(\bar{N})}{v'(\bar{N})} = \epsilon_{\bar{N}} \leq \frac{1}{4},$$

– a contradiction again.  $\square$

Note that this property implies each feasible set contains at most 3 items. In the following we reconsider (slight variants of) two packing heuristics that have been introduced earlier in [2] resp. [13]. The first one, which we call Item Packing, seeks to first pack large items, then small ones on top of these, without regarding the optimum fractional solution. The second one, which we call Set Packing, starts out from the optimum fractional solution  $y'$  and seeks to extract an integer packing based on the feasible sets that are (fractionally) packed by  $y'$ .

We first deal with Item Packing. Call an item  $i$  *large* if  $a_i > 1/3$  and *small* otherwise. Let  $L$  and  $S$  denote the set of large resp. small items. If no misunderstanding is possible, we will also interpret  $L$  as the game  $N$  restricted to the large items (and  $k$  bins). Note that at most two large items fit into a bin. Thus packing  $L$  is basically a matching problem. This is why we can easily solve it to optimality and why the gap is fairly small (just like in the example from section 2). More precisely, Theorem 3.2 from [7] can be stated as

**Lemma 3 ([7]).**  $gap(L) = v'(L) - v(L) \leq \frac{1}{4}$ .

This motivates the following Item Packing heuristic:

### Item Packing

- Compute an optimum integral packing  $\hat{y}$  for  $L$ .
- Try to add as many small items on top of  $\hat{y}$  as possible.

There is no need to specify how exactly small items are added in step 2. A straightforward way would be to sort the small items non-increasingly and apply some first or best fit heuristic. Let  $\hat{F}_1, \dots, \hat{F}_k$  denote the feasible sets in the integral packing  $\hat{y}$  produced by Item Packing, i.e.,

$$\text{Output Item Packing: } \hat{y} \equiv (\hat{F}_1, \dots, \hat{F}_k). \quad (3.2)$$

Note that, due to Lemma 3, Item Packing performs quite well w.r.t. the large items. Thus we expect Item Packing to perform rather well in case there are only a few small items or, more generally, if Item Packing leaves only a few small items unpacked. Let  $S_1 \subseteq S$  denote the set of small items that get packed in step 2 and let  $S_2 := S \setminus S_1$  be the set of unpacked items. We can then prove the following bounds:

**Lemma 4.**  $a_{\hat{F}_j} > \frac{2}{3}$  for all  $j = 1, \dots, k$ . Hence,  $v(N) > \frac{2}{3}k$  and  $v'(N) > \frac{8}{9}k$ .

*Proof.* By definition, the first step of Item Packing produces an optimum integral packing of  $L$  of value  $v(L)$ . Thus the final outcome  $\hat{y}$  has value  $w(\hat{y}) = v(L) + a_{S_1}$ . Hence  $v(N) \geq v(L) + a_{S_1}$ . The fractional value, on the other hand, is clearly bounded by  $v'(N) \leq v'(L) + a_S$ . Thus in case  $S_2 = \emptyset$  (i.e.,  $S_1 = S$ ) we find

$$v'(N) - v(N) \leq v'(L) - v(L) \leq 1/4,$$

implying that

$$\epsilon_N = \frac{v'(N) - v(N)}{v'(N)} \leq \frac{1/4}{k/2} \leq \frac{1}{4} \quad \text{for } k \geq 2,$$

contradicting our assumption that  $1/4\text{-core}(N) = \emptyset$ . (Note that for  $k = 1$  we always have  $v = v'$ , i.e.,  $\epsilon_N = 0$ , as remarked earlier in section 2.)

Thus we conclude that  $S_2 \neq \emptyset$ . But this means that the packing  $\hat{y}$  produced by Item Packing fills each bin to more than  $2/3$ , i.e.,  $a_{\hat{F}_j} > 2/3$  for all  $j$  (otherwise, if  $a_{\hat{F}_j} \leq 2/3$ , any item in  $S$  could be packed on top of  $\hat{F}_j$ ). Hence  $v(N) \geq w(\hat{y}) = \sum_j a_{\hat{F}_j} > \frac{2}{3}k$ . Furthermore, due to our assumption that  $\epsilon_N > 1/4$ , we know that  $v'(N) > \frac{4}{3}v(N) > \frac{8}{9}k$ .  $\square$

As we have seen in the proof of Lemma 4, we may assume  $S_2 \neq \emptyset$ . The following result strengthens this observation:

**Lemma 5.**  $|S_2| \geq \frac{2}{3}k - \frac{3}{4}$ .

*Proof.* As in the proof of Lemma 4 we find

$$\begin{aligned} v'(N) &\leq v'(L) + a_{S_1} + a_{S_2}, \\ v(N) &\geq v(L) + a_{S_1}. \end{aligned}$$

Thus, using Lemma 3, we get

$$v'(N) - v(N) \leq a_{S_2} + \frac{1}{4} \leq \frac{|S_2|}{3} + \frac{1}{4}. \quad (3.3)$$

On the other hand,  $\epsilon_N > 1/4$  and  $v'(N) > \frac{8}{9}k$  (Lemma 4) imply

$$v'(N) - v(N) = \epsilon_N v'(N) > \frac{2}{9}k. \quad (3.4)$$

Combining (3.3) and (3.4), the bound on  $|S_2|$  follows.  $\square$

Thus we are left to deal with instances where Item Packing leaves a considerable amount  $|S_2|$  of small items unpacked. As it turns out, a completely different kind of packing heuristic, so-called Greedy Selection, first analyzed in [13] is helpful in this case. The basic idea is simple: We start with an optimum fractional packing  $y'$  of value  $w(y') = v'(N)$ . Let  $F'_1, \dots, F'_m$  denote the feasible sets in the support of  $y'$  (i.e., those that are fractionally packed) and assume that  $a_{F'_1} \geq \dots \geq a_{F'_m}$ . Greedy Selection starts with  $F_1 := F'_1$  and then, after having selected  $F_1, \dots, F_{j-1}$ , defines  $F_j$  to be the first set in the sequence  $F'_1, \dots, F'_m$  that is disjoint from  $F_1 \cup \dots \cup F_{j-1}$ . It is not difficult to see (*cf.* below) that we can select disjoint feasible sets  $F_1, \dots, F_r$  with  $r = \lceil k/3 \rceil$  in this way.

### Greedy Selection

(Input: Opt. fractional  $y'$  with  $\text{supp } y' \hat{=} \{F'_1, \dots, F'_m\}$ ,  $a_{F'_1} \geq \dots \geq a_{F'_m}$ )

Initiate:  $\mathcal{F}' := \{F'_1, \dots, F'_m\}$

FOR  $j = 1$  to  $r = \lceil k/3 \rceil$  DO

  ·  $F_j \leftarrow$  first set in the list  $\mathcal{F}'_j$

  ·  $\mathcal{F}'_{j+1} \leftarrow \mathcal{F}' \setminus \{F'_t \mid F'_t \cap F_j \neq \emptyset\}$

End

Note that Greedy Selection starts with  $\mathcal{F}' = \text{supp } y'$ , a system of feasible sets of total length  $y'_{\mathcal{F}'} := \sum_{F' \in \mathcal{F}'} y'_{F'} = k$ . In each step, since  $|F_j| \leq 3$ , we remove feasible sets  $F'_t$  of total length  $\sum_{F'_t \in \mathcal{F}'_j, F'_t \cap F_j \neq \emptyset} y'_{F'_t}$  bounded by

$$\sum_{\substack{F'_t \in \mathcal{F}'_j \\ F'_t \cap F_j \neq \emptyset}} y'_{F'_t} \leq \sum_{i \in F_j} \sum_{\substack{F'_t \in \mathcal{F}' \\ F'_t \ni i}} y'_{F'_t} \leq \sum_{i \in F_j} 1 \leq 3.$$

Actually, the upper bound  $\leq 3$  is strict since  $F'_t = F_j$  is counted  $|F_j|$  times (once for each  $i \in F_j$ ).

Thus, in each step we remove feasible sets of total length at most 3, so we certainly can continue the construction for  $k = \lceil k/3 \rceil$  steps.

**Lemma 6.** *Greedy Selection constructs feasible sets  $F_1, \dots, F_r \in \mathcal{F}'$  with  $r = \lceil k/3 \rceil = \frac{1}{3}(k+s)$ ,  $s \equiv -k \pmod{3}$ , of total value  $a_{F_1} + \dots + a_{F_r} \geq \frac{1}{3}(v'(N) + \frac{2}{3}s)$ .*

*Proof.* The first part has been argued already above. (Note that if we write  $r = \frac{1}{3}(k+s)$ , then  $s = 3r - k$ , so  $s \equiv -k \pmod{3}$ .) To prove the lower bound, we first prove

*Claim 1:*  $a_{F_r} > \frac{2}{3}$ .

*Proof of Claim 1.* Assume to the contrary that  $a_{F_r} \leq 2/3$ . In the constructive process of Greedy Selection, when we have selected  $F_1, \dots, F_{r-1}$ , the remaining feasible set system  $\mathcal{F}'$  has still length  $\geq k - 3(r-1) = 3 - s \geq 1$ . As  $F_r$  has maximum size (value) among all sets in  $\mathcal{F}'$ , we know that each set in  $\mathcal{F}'$  has size  $\leq 2/3$ . Decrease  $y'$  on  $\mathcal{F}'$  arbitrarily such that the resulting fractional packing  $\tilde{y}'$  has total length  $\sum_F \tilde{y}'_F = k - 1$ . By construction, the corresponding game  $\tilde{N}$  with one bin removed has fractional value

$$v'(\tilde{N}) \geq w(\tilde{y}') \geq w(y') - \frac{2}{3} = v'(N) - \frac{2}{3}.$$

By minimality of our counterexample  $N$ , the game  $\tilde{N}$  admits an integral packing  $\tilde{y}$  of value

$$v(\tilde{N}) = w(\tilde{y}) \geq \frac{3}{4}v'(\tilde{N}) \geq \frac{3}{4}(v'(N) - \frac{2}{3}) = \frac{3}{4}v'(N) - \frac{1}{2}.$$

Extending this packing  $\tilde{y}$  by filling the  $k^{\text{th}}$  bin to at least  $1/2$  in a simple greedy manner (order not yet packed items non-increasingly in size and try to pack them into bin  $k$  in this order) yields an integral packing  $y$  for  $N$  of value

$$w(y) \geq w(\tilde{y}) + \frac{1}{2} \geq \frac{3}{4}v'(N),$$

contrary to our assumption on  $N$ . This finishes the proof of Claim 1.

To prove the bound on  $a_{F_1} + \dots + a_{F_r}$  in Lemma 6, let  $\mathcal{R}'_j \subseteq \mathcal{F}'_j$  denote those feasible sets that are removed from  $\mathcal{F}'_j$  in step  $j$ , i.e.,  $\mathcal{R}'_j = \mathcal{F}'_j \setminus \mathcal{F}'_{j+1}$ . As we have seen,  $\mathcal{R}'_j$  has total length  $y'_{\mathcal{R}'_j} = \sum_{F' \in \mathcal{R}'_j} y'_{F'} \leq 3$ . Assume that we, in addition, also decrease the  $y'$ -value on the least valuable sets in  $\mathcal{F}'_r$  by a total of  $3 - y'_{\mathcal{R}'_j}$  in each step. Thus we actually decrease the length of  $y'$  by exactly 3 in each step without any further impact on the construction. The total amount of value removed this way in step  $j$  is bounded by  $3a_{F_j}$ . If  $k \equiv 0 \pmod{3}$ , we remove in all  $r = k/3$  rounds exactly  $v'(N)$ . Thus

$$v'(N) \leq \sum_{j=1}^r 3a_{F_j},$$

as claimed.

When  $k \not\equiv 0 \pmod{3}$ , the same procedure yields a reduced length of  $\mathcal{F}'_r$  after  $r-1$  steps, namely  $y'_{\mathcal{F}'_r} = k - 3(r-1) = 3 - s$ . So in the last step we remove a set

$\mathcal{R}'_r$  of value at most  $(3-s)a_{F_r} \geq \frac{2}{3}(3-s) = 2 - \frac{2}{3}s$  in the last step. Summarizing, the total value removed is

$$\begin{aligned} v'(N) &\leq 3a_{F_1} + \cdots + 3a_{F_{r-1}} + (3-s)a_{F_r} \\ &= 3(a_{F_1} + \cdots + a_{F_r}) - sa_{F_r} \\ &\leq 3(a_{F_1} + \cdots + a_{F_r}) - \frac{2}{3}s, \end{aligned}$$

proving the claim.  $\square$

A natural idea is to extend the greedy selection  $F_1, \dots, F_r$  in a straightforward manner to an integral packing  $y \hat{=} (F_1, \dots, F_k)$ , where  $F_{r+1}, \dots, F_k$  are determined by applying Item Packing to the remaining items and remaining  $k-r$  bins. However, this does not necessarily yield a sufficiently high packing value  $w(y)$ : Indeed, it may happen that the remaining  $k-r \approx \frac{2}{3}k$  bins get only filled to roughly  $1/2$ , so the total packing value is approximately  $w(y) \approx \frac{1}{3}v'(N) + \frac{1}{2} \cdot \frac{2}{3}k$ , which equals  $\frac{2}{3}v'(N)$  in case  $v'(N) \approx k$ .

However, the estimate  $a_{F_{r+1}} + \cdots + a_{F_k} \approx \frac{1}{2}(k-r)$  is rather pessimistic. In particular, if we assume that the packing  $y$  (obtained by Greedy Selection plus Item Packing the remaining  $k-r$  bins) leaves some small items unpacked, then each of the  $k-r$  bins must necessarily be filled to at least  $2/3$  (otherwise any small item could be added on top of  $y$ ). This would yield

$$w(y) \approx \frac{1}{3}v'(N) + \frac{2}{3}k \cdot \frac{2}{3} \geq \left(\frac{1}{3} + \frac{4}{9}\right)k \geq \frac{7}{9}k \geq \frac{7}{9}v'(N),$$

which is certainly sufficient for our purposes. (We do not use this estimate later on: It is only meant to guide our intuition.) Thus the crucial instances are those where  $y$  packs all small items – and hence does not pack all large items. Thus, when Item Packing is performed on the  $k-r$  remaining bins, the first phase will pack large items into these  $k-r$  bins until each bin is at least filled to  $1/2$ . Packing small items on top of any such bin would result in a bin filled to at least  $3/4$ . Thus, again, the worst case appears to happen when all small items (and there are quite a few of these, *cf.* Lemma 5) are already contained in  $F_1 \cup \cdots \cup F_r$ . Assume for a moment that each of  $F_1, \dots, F_r$  ( $r \approx k/3$ ) contains two of the small items in  $S_2$ . (Recall that  $|S_2| \gtrsim \frac{2}{3}k$ .) Then each of  $F_1, \dots, F_r$  can contain only one other item in addition. Thus about  $\frac{2}{3}k$  of the sets  $\hat{F}_j$  computed by Item Packing must be disjoint from  $F_1 \cup \cdots \cup F_r$  (as no  $\hat{F}_j$  contains any item from  $S_2$ ). Thus we could extend  $F_1, \dots, F_r$  by roughly  $\frac{2}{3}k \approx k-r$  sets, say,  $\hat{F}_{r+1}, \dots, \hat{F}_k$  from Item Packing to obtain a packing  $(F_1, \dots, F_r, \hat{F}_{r+1}, \dots, \hat{F}_k)$  of value  $\geq \frac{1}{3}v'(N) + \frac{2}{3}k \cdot \frac{2}{3}$  (as each  $\hat{F}_j$  has size  $\geq \frac{2}{3}$ ), which is again enough.

In general, the Greedy Selection  $F_1, \dots, F_r$  will contain some – but not all – of  $S_2$  and we will have to work out a trade-off between the two extreme cases considered above: Let

$$\gamma := |S_2 \cap (F_1 \cup \cdots \cup F_r)|.$$

Thus  $F_1 \cup \cdots \cup F_r$  contains at most  $3r-\gamma$  items that are not in  $S_2$  – and hence possibly contained in some  $\hat{F}_j$ . So there are at most  $3r-\gamma$  different  $\hat{F}_j$  that intersect  $F_1 \cup \cdots \cup F_r$  – and hence at least  $k-(3r-\gamma)$  different  $\hat{F}_j$  that do not intersect

$F_1 \cup \dots \cup F_r$ . We add these  $k - 3r + \gamma = \gamma - s$  different  $\hat{F}_j$ , say,  $\hat{F}_{r+1}, \dots, \hat{F}_{r+\gamma-s}$  to  $F_1, \dots, F_r$  to obtain a “partial” packing  $F_1, \dots, F_r, \hat{F}_{r+1}, \dots, \hat{F}_{r+\gamma-s}$ . Here we may assume that  $r + \gamma - s < k$ . Otherwise, we finish with a complete packing  $\bar{y} = F_1, \dots, F_r, \hat{F}_{r+1}, \dots, \hat{F}_k$  of value (use Lemma 6 and  $a_{\hat{F}_j} \geq 2/3$ ).

$$\begin{aligned} w(\bar{y}) &\geq \frac{1}{3}(v'(N) + \frac{2}{3}s) + (k - r) \cdot \frac{2}{3} \\ &= \frac{1}{3}v'(N) + \frac{2}{9}s + (\frac{2}{3}k - \frac{1}{3}s)\frac{2}{3} \\ &= \frac{1}{3}v'(N) + \frac{4}{9}k \\ &\geq (\frac{1}{3} + \frac{4}{9})v'(N) \\ &> \frac{3}{4}v'(N), \end{aligned}$$

contrary to our assumption that  $v(N) < \frac{3}{4}v'(N)$ .

Thus  $r + \gamma - s < k$  holds indeed and therefore we may complete our partial packing  $F_1, \dots, F_r, \hat{F}_{r+1}, \dots, \hat{F}_{r+\gamma-s}$  by Item Packing the remaining items to the remaining  $k - (r + \gamma - s)$  bins, resulting in an integral packing

$$\bar{y} \hat{=} F_1, \dots, F_r, \hat{F}_{r+1}, \dots, \hat{F}_{r+\gamma-s}, F_{r+\gamma-s+1}, \dots, F_k.$$

This completes the description of our heuristic

### Set Packing

- Get  $F_1, \dots, F_r$  from Greedy Selection and let  $\gamma := |S_2 \cap (F_1 \cup \dots \cup F_r)|$
- Extend with sets  $\hat{F}_j$  from Item Packing to  $F_1, \dots, F_r, \hat{F}_{r+1}, \dots, \hat{F}_{r+\gamma-s}$
- Complete by Item Packing to  $F_1, \dots, F_r, \hat{F}_{r+1}, \dots, \hat{F}_{r+\gamma-s}, F_{r+\gamma-s+1}, \dots, F_k$ .

**Lemma 7.** *Set Packing packs all of  $S$  (and hence not all of  $L$ ).*

*Proof.* If Set Packing leaves some small item unpacked, then necessarily  $a_{F_j} > 2/3$  for  $j = r + \gamma - s + 1, \dots, k$ . Thus the same computation as above yields

$$w(\bar{y}) \geq \frac{1}{3}(v'(N) + \frac{2}{3}s) + (k - r)\frac{2}{3} > \frac{3}{4}v'(N),$$

a contradiction. Thus all of  $S$  gets packed. If, in addition, all of  $L$  would get packed, then the value of the resulting packing  $\bar{y}$  were  $w(\bar{y}) \geq a_S + a_L \geq v'(N)$ , a contradiction again.  $\square$

Thus, in phase 3 of Set Packing, when we apply Item Packing to the last  $k - (r + \gamma - s)$  bins, each bin gets first filled to at least  $1/2$  by large items, and then (possibly among other small items), the remaining  $|S_2| - \gamma$  items from  $S_2$  get packed on top of (some of) the last  $k - (r + \gamma - s)$  bins. So these last  $k - (r + \gamma - s)$  bins contribute at least

$$\frac{1}{2}(k - (r + \gamma - s)) + \frac{1}{4}(|S_2| - \gamma)$$

to the total value of  $\bar{y}$ .

Summarizing,  $w(\bar{y})$  can be estimated as

$$\begin{aligned} w(\bar{y}) &\geq \frac{1}{3}(v'(N) + \frac{2}{3}s) + (\gamma - s) \cdot \frac{2}{3} \\ &\quad + \frac{1}{2}(k - (r + \gamma - s)) + \frac{1}{4}(|S_2| - \gamma). \end{aligned} \tag{3.5}$$

In case  $k \equiv 0 \pmod{3}$ , we have  $s = 0$ ,  $r = k/3$  and  $|S_2| \geq \frac{2}{3}k$  (by Lemma 5). So (3.5) simplifies to

$$w(\bar{y}) \geq \frac{1}{3}v'(N) + \frac{1}{2} \cdot \frac{2}{3}k + \frac{1}{4} \cdot \frac{2}{3}k - \frac{1}{12}\gamma \geq \frac{3}{4}v'(N) + \frac{1}{12}k - \frac{1}{12}\gamma,$$

proving that  $w(\bar{y}) \geq \frac{3}{4}v'(N)$  if  $\gamma \leq k$ . But this is true: Indeed, as we have already argued above, we may even assume that  $r + \gamma - s = r + \gamma < k$ . (Recall that we consider the case  $s = 0$  here.) In case  $k \not\equiv 0 \pmod{3}$ , by similar computation, we can show that  $w(\bar{y}) \geq \frac{3}{4}v'(N)$  still holds.

Summarizing, both cases ( $k \equiv 0$  and  $k \not\equiv 0 \pmod{3}$ ) yield that  $N$  cannot be a counterexample, so we have proved

**Theorem 8**  $\epsilon_N \leq 1/4$  for all instances of the bin packing game.

## 4 Remarks and Open Problems

We like to note that – even though our proof in section 3 is indirect – it can easily be turned into a constructive proof. Indeed, we implicitly show that either Item Packing or Set Packing yields an integral packing  $y$  of value  $w(y) \geq \frac{3}{4}v'(N)$ . Also note that an optimum fractional packing  $y'$  (as input to Greedy Selection) is efficiently computable: Indeed, as any feasible set may contain at most 3 items, the total number of feasible sets is  $O(n^3)$ , so the LP for computing  $y'$  has polynomial size. Thus, in particular, our approach also yields a strengthening of the result in [3] (efficient 3/4 approximation).

In [13] it was conjectured that  $\epsilon_N \leq 1/7$  is true for all instances. (This bound would be tight as can be seen from the small example presented in section 2.) We do not expect that our arguments provide any clue about how to approach 1/7.

A probably even more challenging conjecture due to Woeginger states that the integrality gap

$$gap(N) = v'(N) - v(N)$$

is bounded by a constant. Until now, this has only been verified for item sizes  $> 1/3$  (cf. Lemma 3). It would be interesting to investigate the case of item sizes  $a_i > 1/4$ . The largest gap found (cf. [12]) so far is  $gap(N) \approx 1/3$ , for a game with 6 bins and 18 items.

## References

1. Bilò, V.: On the packing of selfish items. In: Proceedings of the 20th International Conference on Parallel and Distributed Processing, IPDPS 2006, p. 45. IEEE Computer Society, Washington, DC (2006)
2. Caprara, A., Kellerer, H., Pferschy, U.: The multiple subset sum problem. SIAM Journal of Optimization 11, 308–319 (1998)
3. Caprara, A., Kellerer, H., Pferschy, U.: A 3/4-approximation algorithm for multiple subset sum. Journal of Heuristics 9(2), 99–111 (2003)
4. Epstein, L., Kleiman, E.: Selfish bin packing. Algorithmica 60, 368–394 (2011)
5. Faigle, U., Fekete, S., Hochstättler, W., Kern, W.: On approximately fair cost allocation in euclidean tsp games. Operations Research Spektrum 20, 29–37 (1998)
6. Faigle, U., Kern, W.: On some approximately balanced combinatorial cooperative games. Methods and Models of Operation Research 38, 141–152 (1993)
7. Faigle, U., Kern, W.: Approximate core allocation for binpacking games. SIAM J. Discrete Math. 11, 387–399 (1998)
8. Goemans, M.X., Skutella, M.: Cooperative facility location games. Journal of Algorithms 50(2), 194–214 (2004)
9. Han, X., Chin, F.Y.L., Ting, H.-F., Zhang, G., Zhang, Y.: A new upper bound 2.5545 on 2d online bin packing. ACM Trans. Algorithms 7(4), 50:1–50:18 (2011)
10. Jain, K., Mahdian, M., Markakis, E., Saberi, A., Vazirani, V.V.: Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. J. ACM 50(6), 795–824 (2003)
11. Jain, K., Vazirani, V.: Applications of approximation algorithms to cooperative games. In: Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing, STOC 2001, pp. 364–372. ACM, New York (2001)
12. Joosten, B.: Relaxation of 3-partition instances. Master's thesis, Radboud University (December 2011)
13. Kern, W., Qiu, X.: Integrality gap analysis for bin packing games. Operations Research Letters 40(5), 360–363 (2012)
14. Kern, W., Qiu, X.: Note on non-uniform bin packing games. Discrete Applied Mathematics (2012) (in press)
15. Kuipers, J.: Bin packing games. Mathematical Methods of Operations Research 47, 499–510 (1998)
16. Liu, Z.: Complexity of core allocation for the bin packing game. Operations Research Letters 37(4), 225–229 (2009)
17. Woeginger, G.J.: On the rate of taxation in a cooperative bin packing game. Mathematical Methods of Operations Research 42, 313–324 (1995)
18. Yu, G., Zhang, G.: Bin packing of selfish items. In: Papadimitriou, C., Zhang, S. (eds.) WINE 2008. LNCS, vol. 5385, pp. 446–453. Springer, Heidelberg (2008)

# On the Advice Complexity of the Online $L(2, 1)$ -Coloring Problem on Paths and Cycles\*

Maria Paola Bianchi<sup>1</sup>, Hans-Joachim Böckenhauer<sup>2</sup>, Juraj Hromkovič<sup>2</sup>,  
Sacha Krug<sup>2</sup>, and Björn Steffen<sup>2</sup>

<sup>1</sup> Dipartimento di Informatica, Università degli Studi di Milano, Italy

<sup>2</sup> Department of Computer Science, ETH Zurich, Switzerland

[maria.bianchi@unimi.it](mailto:maria.bianchi@unimi.it),

[{hjb,juraj.hromkovic,sacha.krug,bjoern.steffen}@inf.ethz.ch](mailto:{hjb,juraj.hromkovic,sacha.krug,bjoern.steffen}@inf.ethz.ch)

**Abstract.** In an  $L(2, 1)$ -coloring of a graph, the vertices are colored with colors from an ordered set such that neighboring vertices get colors that have distance at least 2 and vertices at distance 2 in the graph get different colors. We consider the problem of finding an  $L(2, 1)$ -coloring using a minimum range of colors in an online setting where the vertices arrive in consecutive time steps together with information about their neighbors and vertices at distance two among the previously revealed vertices. For this, we restrict our attention to paths and cycles.

Offline, paths can easily be colored within the range  $\{0, \dots, 4\}$  of colors. We prove that, considering deterministic algorithms in an online setting, the range  $\{0, \dots, 6\}$  is necessary and sufficient while a simple greedy strategy needs range  $\{0, \dots, 7\}$ .

Advice complexity is a recently developed framework to measure the complexity of online problems. The idea is to measure how many bits of advice about the yet unknown parts of the input an online algorithm needs to compute a solution of a certain quality. We show a sharp threshold on the advice complexity of the online  $L(2, 1)$ -coloring problem on paths and cycles. While achieving color range  $\{0, \dots, 6\}$  does not need any advice, improving over this requires a number of advice bits that is linear in the size of the input. Thus, the  $L(2, 1)$ -coloring problem is the first known example of an online problem for which sublinear advice does not help.

We further use our advice complexity results to prove that no randomized online algorithm can achieve a better expected competitive ratio than  $\frac{5}{4}(1 - \delta)$ , for any  $\delta > 0$ .

## 1 Introduction

Graph coloring is a well-known problem with many applications and many variants of it have been considered in the literature. One of these variations is motivated by a problem arising in the context of assigning frequencies to transmitters in a multihop radio network. To avoid interference, the difference between the

---

\* This work was partially supported by SNF grant 200021-141089.

frequencies used by transmitters should be anti-proportional to their proximity, see the survey by Murphey et al. [17] for an overview of frequency assignment problems. The simplest graph-theoretic model of the frequency assignment problem is the  $L(2, 1)$ -coloring problem, which was introduced by Griggs and Yeh [12]. It is also commonly known as the radio coloring problem [8]. Here, the transmitters are the vertices of a graph and the frequencies are modeled by colors from an ordered set, usually  $\{0, 1, \dots, \lambda\}$ , for some  $\lambda \in \mathbb{N}$ . For the coloring, two classes of proximity are distinguished. Neighboring vertices have to be assigned colors that are at least 2 apart in their given order, and vertices at distance 2 (called *square neighbors* in the following) have to get different colors. The goal is to find such a coloring of the graph with a minimum  $\lambda$ . This problem has been intensively studied for various graph classes, for a survey, see the papers by Yeh [21] or Bodlaender et al. [2]. In this paper, we introduce and investigate an online version of the problem. Here, the vertices of the graph appear in consecutive time steps together with information about those neighbors and square neighbors that have already been revealed. In each time step, the online algorithm has to irrevocably determine the color of the newly revealed vertex respecting the above restrictions.

The quality of an online solution is traditionally measured using the competitive analysis introduced by Sleator and Tarjan [20]. The *competitive ratio*, defined as the quotient of the cost of the computed online solution and the cost of an optimal (offline) solution, is the standard measure for the quality of an online algorithm. A detailed introduction to the theory of online algorithms can, e.g., be found in the textbook by Borodin and El-Yaniv [7]. The recently introduced framework of *advice complexity* aims at a more fine-grained measurement of the complexity of online problems. The idea is to measure how much information about the future parts of the input is needed to compute an optimal solution or a solution with a specific competitive ratio [13]. This is modeled by an oracle that knows the whole input in advance and prepares some infinite tape of *advice bits* which the online algorithm can use as an additional resource. The *advice complexity* of an online algorithm is then defined as the maximum length of the prefix of the advice tape that the algorithm accesses over all inputs of a fixed length. The advice complexity of an online problem is the minimal advice complexity over all admissible algorithms. The first model of advice complexity was introduced by Dobrev et al. [9] and later refined by Emek et al. [10]. The first model was not exact enough, discrepancies up to a multiplicative factor were possible. The latter model was suitable only if the advice complexity was at least linear. We are using the general model as proposed by Hromkovič et al. [13]. The concept was successfully applied to various online problems [5,10,4,18,3,15,1,11].

There are many connections between advice complexity and randomized online computations [4,14,6]. Obviously, every online algorithm using  $b$  advice bits is as least as powerful as a randomized algorithm using the same number of random binary decisions. But under certain conditions, one can use lower bounds on the advice complexity also to prove lower bounds on randomized online computation with an unbounded number of random bits [4].

**Table 1.** Upper and lower bounds on the advice complexity of the  $L(2, 1)$ -coloring problem on paths and cycles, where  $d_1$  and  $d_2$  are positive constants. (1.5-competitiveness can be achieved without advice.)

Quality	Lower Bound	Upper Bound
optimal ( $\lambda = 4$ )	$0.0518n$	$0.6955n + d_1$
1.25-competitive ( $\lambda = 5$ )	$3.9402 \cdot 10^{-10}n$	$0.4n + d_2$

In this paper, we focus on the online  $L(2, 1)$ -coloring problem on paths and cycles. We concentrate on presenting the results for paths in this paper. All results can be generalized to cycles and graphs of maximum degree 2, i. e., graphs whose components are paths or cycles.

While the  $L(2, 1)$ -coloring problem on paths is almost trivial in the offline case (simply coloring the vertices along the path using the pattern 0-2-4 is optimal), it turns out to be surprisingly hard in the online case. We first analyze a simple greedy strategy and show that it uses  $\lambda = 7$ . Then we present an improved deterministic online algorithm using  $\lambda = 6$  and prove a matching lower bound for deterministic online algorithms without advice. Considering online algorithms with advice, we prove that, in order to achieve an optimal solution on a path on  $n$  vertices,  $0.6955n + d$  advice bits are sufficient, for some positive constant  $d$ , and  $0.0519n$  advice bits are necessary. Surprisingly, also to compute a 1.25-competitive solution (i. e., for  $\lambda = 5$ ), a linear number of advice bits is necessary. Thus, the  $L(2, 1)$ -coloring is the first known problem for which sublinear advice does not help at all, not even on the very simple graph classes of paths and cycles. Finally, we employ this lower bound to show that no randomized online algorithm for the online  $L(2, 1)$ -coloring problem has an expected competitive ratio of  $\frac{5}{4}(1 - \delta)$ , for any  $\delta > 0$ . Table 1 summarizes our advice complexity results. Due to space limitations, some proofs are omitted.

## 2 Preliminaries

Let us first formally define the framework we are using.

**Definition 1.** Consider an input sequence  $I = (x_1, \dots, x_n)$  for some minimization problem  $U$ . An online algorithm  $A$  computes the output sequence  $A(I) = (y_1, \dots, y_n)$ , where  $y_i = f(x_1, \dots, x_i)$  for some function  $f$ . The cost of the solution is denoted by  $\text{cost}(A(I))$ . An algorithm  $A$  is strictly  $c$ -competitive, for some  $c \geq 1$ , if, for every input sequence  $I$ ,  $\text{cost}(A(I)) \leq c \cdot \text{cost}(\text{Opt}(I))$ , where  $\text{Opt}$  is an optimal offline algorithm for the problem.  $A$  is optimal if it is strictly 1-competitive.

Because we are dealing with a problem whose solution costs are bounded by a constant, we only consider strict competitiveness in this paper and hence omit the term “strictly”.

**Definition 2.** Consider an input sequence  $I = (x_1, \dots, x_n)$ . An online algorithm  $A$  with advice computes the output sequence  $A^\phi(I) = (y_1, \dots, y_n)$  such that  $y_i$  is computed from  $\phi, x_1, \dots, x_i$ , where  $\phi$  is the content of the advice tape, i.e., an infinite bit string. The algorithm  $A$  is  $c$ -competitive with advice complexity  $s(n)$  if, for every  $n$  and every input sequence  $I$  of length at most  $n$ , there is some  $\phi$  such that  $\text{cost}(A^\phi(I)) \leq c \cdot \text{cost}(\text{Opt}(I))$  and at most the first  $s(n)$  bits of  $\phi$  have been accessed during the computation of  $A$  on  $I$ .

Let  $G = (V, E)$  be a graph with vertex set  $V$  and edge set  $E$ . The number of vertices in  $G$  is denoted by  $n$ . A *path* is a graph  $P = (V, E)$  such that  $V = \{v_1, v_2, \dots, v_n\}$  and  $E = \{\{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_{n-1}, v_n\}\}$ . For short, we write  $P = (v_1, v_2, \dots, v_n)$  and  $v_i \in P$ . A *cycle* is a path with an edge between the first and the last vertex, i.e.,  $\{v_1, v_n\} \in P$ . The *length* of a path is the number of edges in it. A *subpath*  $(v_i, v_{i+1}, \dots, v_j)$ , for  $1 \leq i \leq j \leq n$ , is the induced subgraph  $P[\{v_i, v_{i+1}, \dots, v_j\}]$ . If two vertices have distance 2 from each other, we call them *square neighbors*, since they are neighbors in  $P^2$ , the graph resulting from adding an edge between any two endpoints of a subpath of length 2. A vertex is *isolated* in some induced subgraph of  $P$  if it has neither direct nor square neighbors.

**Definition 3.** An  $L(2, 1)$ -coloring of a graph  $G$  is a function assigning to every vertex of  $G$  a color from the set  $\{0, 1, \dots, \lambda\}$  such that adjacent vertices receive colors at least 2 apart and square neighbors receive different colors, i.e., at least 1 apart.

The aim of the  $L(2, 1)$ -coloring problem is to find an  $L(2, 1)$ -coloring minimizing  $\lambda$ . An  $L(2, 1)$ -coloring of a graph with minimal  $\lambda$  is called optimal.

Clearly,  $\lambda = 4$  is enough to color any path optimally offline. The algorithm simply follows the pattern 0-2-4 periodically from left to right.

**Lemma 1 (Griggs and Yeh [12]).** To color a path on at least five vertices or any cycle,  $\lambda = 4$  is necessary and sufficient.  $\square$

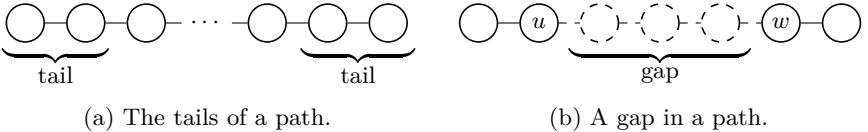
In the following, we always mean  $L(2, 1)$ -coloring when we speak of coloring.

**Definition 4.** The online  $L(2, 1)$ -coloring problem is the following minimization problem. For some graph  $G = (V, E)$  and an order on  $V$ , the vertices in  $V$  are revealed one by one with respect to this order. Let  $V_t$  denote the set of vertices revealed up to time step  $t$ . Together with each revealed vertex  $v$ , all edges between  $v$  and previously revealed vertices are revealed as well, i.e., up to time step  $t$ , the graph  $G_t = G[V_t]$  is revealed. Additionally, in every time step  $t$ , information about the square neighbors of  $v$  among the vertices in  $V_t$  is revealed.<sup>1</sup>

The goal is to find an  $L(2, 1)$ -coloring  $c: V \rightarrow \{0, \dots, \lambda\}$  minimizing  $\lambda$ . For each revealed vertex  $v$ , the online algorithm immediately has to decide what color  $c(v)$  this vertex gets.

---

<sup>1</sup> Intuitively speaking, the algorithm is told which of the already revealed vertices are square neighbors of  $v$ , but it gets no information about intermediate vertices that are not yet revealed. This additional constraint is usually not part of an online graph coloring setting, but necessary for our problem.



**Fig. 1.** The tails and gaps of a path

In this paper, we consider paths and cycles as inputs for the online  $L(2, 1)$ -coloring problem. Note that the online algorithm neither gets information about the number of vertices in  $G$  nor about the index of the currently revealed vertex, i. e., its position in the graph. In other words, the vertices are anonymous. Otherwise, the problem is trivial. In a path, the algorithm just assigns color  $2 \cdot (i \bmod 3)$  to vertex  $v_i$  and is always optimal. In a cycle, this is also possible, but the algorithm needs to color the last few vertices differently with respect to the value  $n \bmod 3$ .

Given an algorithm  $A$  for the online  $L(2, 1)$ -coloring problem on paths, we denote by  $c_A(P)$  the coloring of the path  $P$  computed by  $A$  and by  $c_A(v)$  the color assigned to the vertex  $v \in P$ . If  $A$  is clear from the context, we only write  $c(P)$  and  $c(v)$ . We call the two directions in which a path/subpath can be extended from a vertex  $v$  the two *sides* of  $v$ . We call the two outmost vertices at the ends of a path/subpath the *tails* (see Figure 1a). If two or three vertices have not yet been revealed between two vertices  $u$  and  $w$ , we call these missing vertices a *gap* (see Figure 1b).

Furthermore, we construct algorithms that sometimes need to choose one color out of three. Since they are reading from a binary advice tape, we need the following lemma.

**Lemma 2 (Seibert et al. [19]).** *Encoding a number  $x$ , for  $1 \leq x \leq 3^n$ , i. e.,  $n$  one-out-of-three decisions, in a bit string needs at most  $\frac{46}{29}n + d$  bits, for some positive constant  $d$ .*  $\square$

Throughout this paper,  $\log x$  denotes the binary logarithm of  $x$ .

### 3 Online Algorithms without Advice

First, we consider the greedy algorithm, i. e., the algorithm that always picks the lowest possible color for a newly revealed vertex. The range  $\{0, 1, \dots, 8\}$ , i. e.,  $\lambda \leq 8$ , is obviously sufficient: A revealed vertex  $v$  has at most two direct and two square neighbors, which together forbid at most eight colors. Thus, at least one of the colors in  $\{0, 1, \dots, 8\}$  is still available for  $v$ . We show, however, that the greedy algorithm never uses the ninth color, i. e., that the range  $\{0, 1, \dots, 7\}$  is sufficient.

**Theorem 1.** *The greedy algorithm for the online  $L(2, 1)$ -coloring problem on paths achieves  $\lambda = 7$ , and this bound is tight.*

Now we show that we cannot ensure optimality without advice. In fact, not even  $\lambda = 5$  is enough to color every path online.

**Theorem 2.** *Without advice,  $\lambda \geq 6$  is necessary to solve the online  $L(2, 1)$ -coloring problem on paths.*

*Proof.* We show that  $\lambda = 5$  is not sufficient.

Let  $A$  be an online algorithm for the online  $L(2, 1)$ -coloring problem on paths using only the colors 0 to 5, and consider the following instance. Seven vertices are revealed isolated, thus defining seven disjoint components  $C_1, C_2, \dots, C_7$ .<sup>2</sup> For  $1 \leq i \leq 7$ , the next revealed vertex is then a direct neighbor of the last revealed vertex in  $C_i$ , until one tail of  $C_i$  is colored either  $0-\alpha$  or  $5-\beta$ , with  $\alpha \in \{2, 3, 4\}$ ,  $\beta \in \{1, 2, 3\}$ . The fact that such a tail always appears after a constant number of steps is guaranteed by the lower bound of  $\lambda = 4$  stated in Lemma 1, because at some point, without loss of generality, a vertex  $v$  is colored 0. The next vertex is either colored 2, 3, or 4, and we are done, or it is colored 5, and we add another vertex, which can only be colored 1, 2, or 3, since its neighbor is colored 5 and its square neighbor is colored 0. (The case where  $v$  is colored 5 is analogous.) Furthermore, the adversary knows when this happens, because  $A$  is deterministic.

There are only six different tails with that property and we have seven components, so one tail must occur twice. We consider an instance that fills the gap between those two tails as follows.

Two tails that have both the form 0-2 or both the form 0-4 are connected by adding two vertices in between. As the two vertices in the gap need to be assigned colors with distance 2, at least one of them receives a color greater than 5. Two tails of the form 0-3 are connected by adding three vertices in between. As the leftmost vertex in the gap receives, without loss of generality, color 1 and the rightmost vertex color 5, the middle vertex can only receive a color greater than 5. (Due to symmetry, the argument is analogous for tails of the form  $5-\beta$ ,  $\beta \in \{1, 2, 3\}$ .)

Thus, none of the gaps can be filled using only the colors 0 to 5.  $\square$

The following theorem shows that this lower bound is tight.

**Theorem 3.** *There is an algorithm that solves the online  $L(2, 1)$ -coloring problem on paths using  $\lambda \leq 6$  without advice.*

## 4 Online Algorithms with Advice

Now, we want to investigate how much advice is necessary and sufficient to achieve optimality and  $5/4$ -competitiveness, i.e., for  $\lambda = 4$  and  $\lambda = 5$ .

---

<sup>2</sup> Note that we did not fix the order of these components nor the distance between them, and so the adversary is free to concatenate the components in any order or direction afterwards.

**Algorithm 1.**


---

```

1: for each revealed vertex  $v$  at time  $t$  do
2:   if  $v$  is isolated in  $G_t$  then
3:     Read a one-out-of-three decision from the advice tape and color  $v$  accord-
       ingly with a color from  $\{0, 2, 4\}$ .
4:   else if  $v$  is connected to one vertex  $w$  that was isolated in  $G_{t-1}$  or  $v$  is
      connected to two vertices  $w$  and  $x$  that were both isolated in  $G_{t-1}$  and have
      distance 3 from each other in  $G_t$  then
5:     Let  $S := \{0, 2, 4\} - c(w)$ .
6:     Read one advice bit  $b$ .
7:     Color  $v$  with the lower of the two colors in  $S$  if  $b = 0$ , and with the higher
       one otherwise.
8:   else
9:     Inspect  $G_t$  to determine the color of  $v$  and color it accordingly.
Output: Color  $c(v)$  of each revealed vertex  $v$ .

```

---

#### 4.1 Lower and Upper Bounds for Optimality

We first show that there is an optimal algorithm that reads advice bits from the tape such that it always knows what color from  $\{0, 2, 4\}$  to assign to the currently revealed vertex. Then, we complement this result by a linear lower bound.

**Theorem 4.** *Algorithm 1 solves the online  $L(2, 1)$ -coloring problem on paths optimally using at most  $0.6955n + d$  advice bits, for some positive constant  $d$ .*

*Proof sketch.* The oracle writes advice bits on the tape in such a way that the final coloring is just a repetition of the pattern 0-2-4.

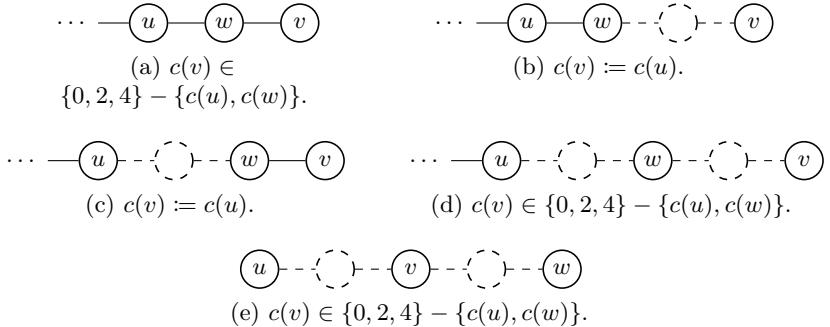
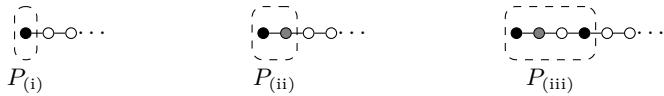
Let us first explain what happens in line 9. There are two cases. In the first case,  $v$  is connected to at least one component  $C$  in  $G_{t-1}$  that consists of more than one vertex. Then,  $c(v)$  is determined by the colors of the vertices in  $C$  by following the pattern 0-2-4 (see Figures 2a to 2d). In the second case,  $v$  is connected to two vertices  $u$  and  $w$  that were isolated in  $G_{t-1}$  and have distance 4 from each other in  $G_t$  as in Figure 2e.

Now we calculate how many advice bits are needed. There are three situations where the algorithm reads advice bits (see Figure 3): (i) an isolated vertex is revealed (one-out-of-three decision); (ii) a vertex connected to one previously isolated vertex is revealed (one advice bit needed); (iii) a vertex is revealed that is connected to two previously isolated vertices that are at distance 3 to each other (one advice bit needed).

Since vertices corresponding to line 9 do not force the algorithm to read advice, we can assume, without loss of generality, that they only occur to connect subpaths  $P_{(i)}$ ,  $P_{(ii)}$  and  $P_{(iii)}$  that enforce situations (i), (ii) or (iii) (see Figure 3). Let  $P'_{(i)}$ ,  $P'_{(ii)}$  and  $P'_{(iii)}$  denote the respective subpaths together with two such connecting vertices.

A careful case analysis now shows that, even for the worst possible input for the algorithm, the claimed amount of advice bits is sufficient.  $\square$

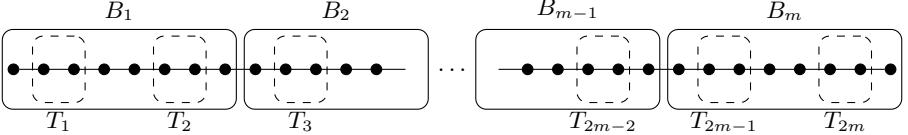
In order to show a lower bound, we need the following technical lemma.

**Fig. 2.** How to color  $v$  in line 9 of Algorithm 1**Fig. 3.** The three types of subpaths forcing Algorithm 1 to read advice bits. The black vertices are revealed isolated to force a one-out-of-three decision and the gray ones require one advice bit to color them.

**Lemma 3.** Consider two paths  $P = (u, v)$  and  $P' = (w, x)$  of two vertices each, and let  $c$  be any valid coloring for  $P$  and  $P'$ . Let now  $(u, v, y, z, w, x)$ ,  $(u, v, y, z, x, w)$ ,  $(v, u, y, z, w, x)$ , and  $(v, u, y, z, x, w)$  be the four possible paths that result from concatenating  $P$  and  $P'$  at arbitrary ends by adding two vertices. For at least one of the four paths, any valid coloring that extends  $c$  needs to use a color greater than 4.

When proving lower bounds on the advice complexity, we usually construct a set of special instances having the property that the algorithm cannot distinguish between their prefixes of a certain length. Using at most  $b$  advice bits, the algorithm can only use  $2^b$  different strategies and thus can only use one of  $2^b$  different colorings. We describe an adversary that constructs at least  $2^b + 1$  different continuations that require pairwise different colorings of the prefix to be colored optimally. Thus, at least one of the instances cannot be colored optimally by the algorithm, i.e.,  $b$  advice bits are not sufficient. Since the prefix of the instance in general consists of some set of isolated vertices and subpaths and the adversary is free to choose the order and orientation of these subgraphs, we cannot in general assume that the prefix consists of exactly the same vertices in the same order for all instances. But for the proof method to work it suffices if the prefixes are isomorphic subgraphs with respect to both direct and square neighbors. We call two subgraphs satisfying this property *indistinguishable*.

**Theorem 5.** Any algorithm that solves the online  $L(2, 1)$ -coloring problem on paths optimally needs to read at least  $0.0518n$  advice bits.



**Fig. 4.** The block-tuple division of the path  $P$  in the proof of Theorem 5

*Proof.* We consider a path  $P$  with  $n = 8m$  vertices, for some  $m \in \mathbb{N}$ , and we partition  $P$  into  $m$  consecutive vertex-disjoint subpaths  $B_1, B_2, \dots, B_m$  of eight vertices each. In every  $B_j = (v_i, v_{i+1}, \dots, v_{i+7})$ , where  $i = 8(j-1)+1$ , we define the two subpaths  $T_{2j-1} = (v_{i+1}, v_{i+2})$  and  $T_{2j} = (v_{i+5}, v_{i+6})$ . We call  $B_j$  the  $j$ -th *block* and  $T_h$  the  $h$ -th *tuple* of  $P$ . This block-tuple division of  $P$  is shown in Figure 4.

We consider as instances all possible online presentations of  $P$  satisfying the following conditions.

- The algorithm first receives the vertices from every tuple  $T_h$ . In  $G_2$ , i. e., after time step 2, only  $T_1$  is revealed, in  $G_4$ ,  $T_1$  and  $T_2$  are revealed, and so on, until all subpaths  $T_1, T_2, \dots, T_{2m}$  have been revealed at time  $4m$ .
- After time step  $4m$ , all remaining vertices are revealed sequentially from left to right.

Under these conditions, there are two possible orders of revealing the vertices in each tuple  $T_h$ . Hence, there are  $4^m$  different instances of this type. Moreover, the prefixes of all instances until time step  $4m$  are indistinguishable, so if two instances get the same advice, they have the same coloring at time step  $4m$ .

Consider an instance  $I$  whose associated advice induces a coloring  $c$  on the  $2m$  tuples in  $G_{4m}$ . We want to determine how many other instances can receive the same advice string, i. e., for how many of them the algorithm can use the same coloring of the tuples in  $G_{4m}$  and still be able to use only colors 0 to 4 in the remaining part. Consider the block  $B_j$  containing the subpaths  $T_{2j-1}$  and  $T_{2j}$ . Lemma 3 shows that there is always an appearance order of the vertices in  $T_{2j-1}$  and  $T_{2j}$  such that the gap in between cannot be colored with values from 0 to 4. This means that  $c$  is suitable for at most three choices of  $B_j$  out of four, and since the same reasoning holds for all other blocks,  $c$  is suitable for at most  $3^m$  different instances.

Hence, there must be at least  $4^m/3^m$  different advice strings, implying that at least  $\log((4/3)^m) = (2 - \log 3) \cdot n/8 \geq 0.0518n$  advice bits are necessary.  $\square$

## 4.2 Lower and Upper Bounds for $\frac{5}{4}$ -Competitiveness

The main idea is to define an algorithm that works like the one in Theorem 3 most of the time and avoids situations that lead to using color 6. The algorithm reads an advice bit for the first vertex revealed on each side, unless it merges two components. Advice bit 0 means follow the pattern 0-3-5, while advice bit 1 means switch to pattern 0-2-4.

**Theorem 6.** *There is an algorithm that solves the online  $L(2, 1)$ -coloring problem on paths with  $\lambda \leq 5$  and uses at most  $0.4n + d$  advice bits, for some positive constant  $d$ .*

**Theorem 7.** *Every algorithm for the online  $L(2, 1)$ -coloring problem on paths with  $\lambda \leq 5$  needs to read at least  $3.9402 \cdot 10^{-10}n$  advice bits.*

*Proof sketch.* Let us fix an arbitrary algorithm  $A$ . In order to force  $A$  to use a certain amount of advice, the adversary constructs an instance as follows.

First, 25 isolated vertices are revealed. Then, there are two possibilities.

1. The adversary selects seven arbitrary vertices  $u_1, u_2, \dots, u_7$  out of the 25. It reveals a neighbor  $v_i$  of each  $u_i$  such that the paths  $(u_i, v_i)$  are still separate components. Then, there are again two possibilities.
  - (a) The adversary connects two arbitrary vertices  $v_j$  and  $v_k$  by adding two or three vertices between them.
  - (b) It reveals another seven vertices  $w_i$  that are neighbors of the  $v_i$  such that the paths  $(u_i, v_i, w_i)$  are still separate components. Then it connects two arbitrary vertices  $w_j$  and  $w_k$  by adding two or three vertices in between.
2. The adversary selects four arbitrary vertices  $u_1, u_2, u_3, u_4$  out of the 25. It reveals four vertices  $v_1, v_2, v_3, v_4$ , two of which form a path between  $u_1$  and  $u_2$  and between  $u_3$  and  $u_4$ , respectively, i.e., there are now two paths  $(u_1, v_1, v_2, u_2)$  and  $(u_3, v_3, v_4, u_4)$ . Then it connects these two paths at arbitrary ends by adding two vertices in between.

At the end, the adversary connects all components according to some fixed order such that the resulting graph is a path.

We are now able to show that, for every coloring of the 25 initial vertices, there is an instance with at most 88 vertices that forces  $A$  to use color 6.

This leads to a number of at most  $N := \binom{25}{7} \cdot 2 \cdot \binom{7}{2} \cdot 2 + \binom{25}{2} \cdot \binom{23}{2} \cdot 4$  problem instances in our special class for a fixed coloring of 25 given initial vertices.

We have shown that, for every coloring of the 25 initial vertices, at least one instance forces a color greater than 5 later on. In other words, one fixed coloring of the initial 25 vertices can be used for at most  $N - 1$  of the  $N$  instances. Because the first 25 vertices are all isolated, reading advice is the only way in which a deterministic algorithm can achieve different colorings of these vertices.

Consider the following scenario. The adversary presents  $m$  times one of the instances described above. There are  $N^m$  possible ways to do this, and every initial coloring of the isolated vertices can result in a valid coloring of the entire graph for at most  $(N-1)^m$  of them. Thus, any algorithm needs to choose between at least  $(N/(N-1))^m$  many colorings for the initial  $25m$  vertices. To distinguish them, it needs at least  $\log((N/(N-1))^m) = m \log(N/(N-1))$  advice bits.

The overall construction consists of  $n \leq 88m + 2(m-1) \leq 90m$  vertices, because we need to connect the instances with each other by adding at least two additional vertices in between. Thus, we need at least  $m \log(N/(N-1)) \geq \frac{n}{90} \log(N/(N-1)) \geq 3.9402 \cdot 10^{-10}n$  advice bits in total.  $\square$

## 5 Randomized Online Algorithms

In this section, we give a lower bound on the competitive ratio achievable by any randomized online algorithm. Our proof is based on the following result.

**Lemma 4 (Böckenhauer et al. [4]).** *Consider an online minimization problem  $U$ , and let  $\mathcal{I}(n)$  be the set of all possible inputs of length  $n$  and  $I(n) := |\mathcal{I}(n)|$ . Furthermore, suppose that there is a randomized online algorithm for  $U$  with worst-case expected competitive ratio at most  $E$ . Then, for any fixed  $\varepsilon > 0$ , it is possible to construct a deterministic online algorithm that uses at most*

$$\log n + 2 \log \log n + \log(\log I(n)/\log(1+\varepsilon)) + c$$

advice bits, for a constant  $c$ ,<sup>3</sup> and achieves a competitive ratio of  $(1+\varepsilon)E$ .  $\square$

Together with this result, Theorem 7 implies that the worst-case expected color range of any randomized online algorithm is bounded from below by a value of almost 5.

**Theorem 8.** *For arbitrarily small  $\delta > 0$ , every randomized algorithm for the online  $L(2, 1)$ -coloring problem on graphs with maximum degree 2 has a worst-case expected competitive ratio of at least  $\frac{5}{4}(1-\delta)$  on sufficiently large instances.*

## 6 Conclusion

We showed that the online  $L(2, 1)$ -coloring problem on graphs consisting only of paths and cycles has the following interesting property. No advice at all is necessary to color a graph with seven colors, but already linear advice is necessary to improve by only one color. In other words, sublinear advice does not help at all. This is something not previously observed—many other problems allow for a smooth tradeoff between advice complexity and competitive ratio.

All our lower bounds directly carry over to more general graph classes that contain paths or cycles as special cases, e.g. trees or Hamiltonian graphs. An open problem is to improve the upper bounds or even match the lower bounds—for paths and cycles as well as for other graph classes.

## References

1. Bianchi, M.P., Böckenhauer, H.-J., Hromkovič, J., Keller, L.: Online coloring of bipartite graphs with and without advice. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 519–530. Springer, Heidelberg (2012)
2. Bodlaender, H.L., Kloks, T., Tan, R.B., van Leeuwen, J.: Approximations for  $\lambda$ -colorings of graphs. Comput. J. 47(2), 193–204 (2004)

---

<sup>3</sup> The (small) constant  $c$  stems from rounding up the logarithms to natural numbers.

3. Böckenhauer, H.-J., Komm, D., Královíč, R., Rossmanith, P.: On the advice complexity of the knapsack problem. In: Fernández-Baca, D. (ed.) LATIN 2012. LNCS, vol. 7256, pp. 61–72. Springer, Heidelberg (2012)
4. Böckenhauer, H.-J., Komm, D., Královíč, R., Královíč, R.: On the advice complexity of the  $k$ -server problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 207–218. Springer, Heidelberg (2011)
5. Böckenhauer, H.-J., Komm, D., Královíč, R., Královíč, R., Mömke, T.: On the advice complexity of online problems. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 331–340. Springer, Heidelberg (2009)
6. Böckenhauer, H.-J., Hromkovič, J., Komm, D., Královíč, R., Rossmanith, P.: On the power of randomness versus advice in online computation. In: Bordihn, H., Kutrib, M., Truthe, B. (eds.) Languages Alive. LNCS, vol. 7300, pp. 30–43. Springer, Heidelberg (2012)
7. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press (1998)
8. Broersma, H.: A General Framework for Coloring Problems: Old Results, New Results, and Open Problems. In: Akiyama, J., Baskoro, E.T., Kano, M. (eds.) IJCCGGT 2003. LNCS, vol. 3330, pp. 65–79. Springer, Heidelberg (2005)
9. Dobrev, S., Královic, R., Pardubská, D.: Measuring the problem-relevant information in input. RAIRO ITA 43(3), 585–613 (2009)
10. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 427–438. Springer, Heidelberg (2009)
11. Forišek, M., Keller, L., Steinová, M.: Advice complexity of online coloring for paths. In: Dedić, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 228–239. Springer, Heidelberg (2012)
12. Griggs, J.R., Yeh, R.K.: Labelling graphs with a condition at distance 2. SIAM J. Discrete Math. 5(4), 586–595 (1992)
13. Hromkovič, J., Královíč, R., Královíč, R.: Information complexity of online problems. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 24–36. Springer, Heidelberg (2010)
14. Komm, D., Královíč, R.: Advice complexity and barely random algorithms. In: Černá, I., Gyimóthy, T., Hromkovič, J., Jefferey, K., Královíč, R., Vukolić, M., Wolf, S. (eds.) SOFSEM 2011. LNCS, vol. 6543, pp. 332–343. Springer, Heidelberg (2011)
15. Komm, D., Královíč, R., Mömke, T.: On the advice complexity of the set cover problem. In: Hirsch, E.A., Karhumäki, J., Lepistö, A., Prilutskii, M. (eds.) CSR 2012. LNCS, vol. 7353, pp. 241–252. Springer, Heidelberg (2012)
16. Komm, D.: Advice and Randomization in Online Computation. PhD Thesis, ETH Zurich (2012)
17. Murphrey, R.A., Pardalos, P.M., Resende, M.G.C.: Frequency assignment problems. In: Du, D.-Z., Pardalos, P.M. (eds.) Handbook of Combinatorial Optimization, Supplement 1, pp. 295–377. Kluwer Academic Publishers (1999)
18. Renault, M.P., Rosén, A.: On online algorithms with advice for the  $k$ -server problem. In: Solis-Oba, R., Persiano, G. (eds.) WAOA 2011. LNCS, vol. 7164, pp. 198–210. Springer, Heidelberg (2012)
19. Seibert, S., Sprock, A., Unger, W.: Advice complexity of the online vertex coloring problem. In: Proc. of CIAC 2013 (to appear, 2013)
20. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communications of the ACM 28(2), 202–208 (1985)
21. Yeh, R.K.: A survey on labeling graphs with a condition at distance two. Discrete Mathematics 306(12), 1217–1231 (2006)

# On Randomized Fictitious Play for Approximating Saddle Points over Convex Sets

Khaled Elbassioni<sup>1</sup>, Kazuhisa Makino<sup>2</sup>, Kurt Mehlhorn<sup>3</sup>,  
and Fahimeh Ramezani<sup>3</sup>

<sup>1</sup> Masdar Institute of Science and Technology, Abu Dhabi, UAE

<sup>2</sup> Graduate School of Information Science and Technology, University of Tokyo,  
Tokyo, 113-8656, Japan

<sup>3</sup> Max Planck Institute for Informatics; Campus E1 4, 66123, Saarbrucken, Germany

**Abstract.** Given two bounded convex sets  $X \subseteq \mathbb{R}^m$  and  $Y \subseteq \mathbb{R}^n$ , specified by membership oracles, and a continuous convex-concave function  $F : X \times Y \rightarrow \mathbb{R}$ , we consider the problem of computing an  $\varepsilon$ -approximate saddle point, that is, a pair  $(x^*, y^*) \in X \times Y$  such that  $\sup_{y \in Y} F(x^*, y) \leq \inf_{x \in X} F(x, y^*) + \varepsilon$ . Grigoriadis and Khachiyan (1995), based on a randomized variant of fictitious play, gave a simple algorithm for computing an  $\varepsilon$ -approximate saddle point for matrix games, that is, when  $F$  is bilinear and the sets  $X$  and  $Y$  are simplices. In this paper, we extend their method to the general case. In particular, we show that, for functions of constant “width”, an  $\varepsilon$ -approximate saddle point can be computed using  $O^*(n + m)$  random samples from log-concave distributions over the convex sets  $X$  and  $Y$ . As a consequence, we obtain a simple randomized polynomial-time algorithm that computes such an approximation faster than known methods for problems with bounded width and when  $\varepsilon \in (0, 1)$  is a fixed, but arbitrarily small constant. Our main tool for achieving this result is the combination of the randomized fictitious play with the recently developed results on sampling from convex sets. A full version of this paper can be found at <http://arxiv.org/abs/1301.5290>.

## 1 Introduction

Let  $X \subseteq \mathbb{R}^m$  and  $Y \subseteq \mathbb{R}^n$  be two *bounded convex* sets. We assume that each set is given by a *membership oracle*, that is an algorithm which given  $x \in \mathbb{R}^m$  (respectively,  $y \in \mathbb{R}^n$ ) determines, in polynomial time in  $m$  (respectively,  $n$ ), whether or not  $x \in X$  (respectively,  $y \in Y$ ). Let  $F : X \times Y \rightarrow \mathbb{R}$  be a continuous convex-concave function, that is,  $F(\cdot, y) : X \rightarrow \mathbb{R}$  is convex for all  $y \in Y$  and  $F(x, \cdot) : Y \rightarrow \mathbb{R}$  is concave for all  $x \in X$ . We assume that we can evaluate  $F$  at rational arguments in constant time. The well-known *saddle-point theorem* (see e.g. [Roc70]) states that

$$v^* = \inf_{x \in X} \sup_{y \in Y} F(x, y) = \sup_{y \in Y} \inf_{x \in X} F(x, y). \quad (1)$$

This can be interpreted as a 2-player zero-sum game, with one player, the minimizer, choosing her/his strategy from a convex domain  $X$ , while the other player,

the maximizer, choosing her/his strategy from a convex domain  $Y$ . For a pair of strategies  $x \in X$  and  $y \in Y$ ,  $F(x, y)$  denotes the corresponding payoff, which is the amount that the minimizer pays to the maximizer. An *equilibrium*, when both  $X$  and  $Y$  are closed, corresponds to a saddle point, which is guaranteed to exist by (1), and the value of the game is the common value  $v^*$ . When an approximate solution suffices or at least one of the sets  $X$  or  $Y$  is open, the appropriate notion is that of  $\varepsilon$ -*optimal strategies*, that a pair of strategies  $(x^*, y^*) \in X \times Y$  such that for a given desired absolute accuracy  $\varepsilon > 0$ ,

$$\sup_{y \in Y} F(x^*, y) \leq \inf_{x \in X} F(x, y^*) + \varepsilon. \quad (2)$$

There is an extensive literature, e.g. [Dan63, Sha58, Roc70], on the existence of saddle points in this class of games and applications of these games. A particularly important case is when the sets  $X$  and  $Y$  are polytopes with an exponential number of facets arising as the convex hulls of combinatorial objects (see full paper for some applications).

One can easily see that (1) can be reformulated as a convex minimization problem over a convex set given by a membership oracle<sup>1</sup>, and hence any algorithm for solving this class of problems, e.g., the Ellipsoid method, can be used to compute a solution to (2), in time polynomial in the input size and  $\text{polylog}(\frac{1}{\varepsilon})$ . However, there has recently been an increasing interest in finding simpler and faster approximation algorithms for convex optimization problems, sacrificing the dependence on  $\varepsilon$  from  $\text{polylog}(\frac{1}{\varepsilon})$  to  $\text{poly}(\frac{1}{\varepsilon})$ , in exchange of efficiency in terms of other input parameters; see e.g. [AHK05, AK07, BBR04, GK92, GK95], [GK98, GK04, Kha04, Kal07, LN93, KY07], [You01, PST91, GK94, GK96, GKPV01], and [Jan04, DJ07].

In this paper, we show that it is possible to get such an algorithm for computing an  $\varepsilon$ -saddle point (2). Our algorithm is based on combining a technique developed by Grigoriadis and Khachiyan [GK95], based on a randomized variant of Brown's fictitious play [Bro51], with the recent results on random sampling from convex sets [LV06, Vem05]. Our algorithm is superior to known methods when the width parameter  $\rho$  (to be defined later) is small and  $\varepsilon \in (0, 1)$  is a fixed but arbitrarily small constant.

## 2 Our Result

We need to make the following technical assumptions:

- (A1) We know  $\xi^0 \in X$ , and  $\eta^0 \in Y$ , and strictly positive numbers  $r_X$ ,  $R_X$ ,  $r_Y$ , and  $R_Y$  such that  $B^m(\xi^0, r_X) \subseteq X \subseteq B^m(\mathbf{0}, R_X)$  and  $B^n(\eta^0, r_Y) \subseteq Y \subseteq B^n(\mathbf{0}, R_Y)$ , where  $B^k(x^0, r) = \{x \in \mathbb{R}^k : \|x - x^0\|_2 \leq r\}$  is the  $k$ -dimensional ball for radius  $r$  centered at  $x^0 \in \mathbb{R}^k$ . In particular, both  $X$  and  $Y$  are full-dimensional in their respective spaces (but maybe open). In what follows we will denote by  $R$  the maximum of  $\{R_X, R_Y, \frac{1}{r_X}, \frac{1}{r_Y}\}$ .

---

<sup>1</sup> Minimize  $F(x)$ , where  $F(x) = \max_y F(x, y)$ .

(A2)  $|F(x, y)| \leq 1$  for all  $x \in X$  and  $y \in Y$ .

Assumption (A1) is standard for algorithms that deal with convex sets defined by membership oracles (see, e.g., [GLS93]), and will be required by the sampling algorithms. Assumption (A2) can be made without loss of generality, since the original game can be converted to an equivalent one satisfying (A2) by scaling the function  $F$  by  $\frac{1}{\rho}$ , where the “width” parameter is defined as  $\rho = \max_{x \in X, y \in Y} |F(x, y)|$ . (For instance, in case of bilinear function, i.e.,  $F(x, y) = xAy$ , where  $A$  is given  $m \times n$  matrix, we have  $\rho = \max_{x \in X, y \in Y} |xAy| \leq \sqrt{mnR_X R_Y} \max\{|a_{ij}| : i \in [m], j \in [n]\}$ .) Replacing  $\varepsilon$  by  $\frac{\varepsilon}{\rho}$ , we get an algorithm that works without assumption (A2) but whose running time is proportional to  $\rho^2$ . We note that such dependence on the width is unavoidable in algorithms that obtain  $\varepsilon$ -approximate solutions for negative functions and whose running time is proportional to  $\text{poly}(\frac{1}{\varepsilon})$  (see e.g. [AHK06, PST91]); otherwise, scaling would yield an exact algorithm for rational inputs. For nonnegative functions  $F$ , the dependence on the width can be avoided, if we consider *relative* approximation errors (see [GK98, Kha04, KY07, You01]).

We assume throughout that  $\varepsilon$  is a positive constant less than 1.

The main contribution of this paper is to extend the randomized fictitious play result in [GK95] to the more general setting given by (2).

**Theorem 1.** *Assume  $X$  and  $Y$  satisfy assumption (A1). Then there is a randomized algorithm that finds a pair of  $\varepsilon$ -optimal strategies in an expected number of  $O(\frac{\rho^2(n+m)}{\varepsilon^2} \ln \frac{R}{\varepsilon})$  iterations, each computing two approximate<sup>2</sup> samples from log-concave distributions. The algorithm<sup>3,4</sup> requires  $O^*(\frac{\rho^2(n+m)^6}{\varepsilon^2} \ln \frac{R}{\varepsilon})$  calls to the membership oracles for  $X$  and  $Y$ .*

When the width is bounded and  $\varepsilon$  is a fixed constant, our algorithm needs  $O^*((n + m)^6 \ln R)$  oracle calls. This is superior to known methods, e.g., the Ellipsoid method, that compute the  $\varepsilon$ -saddle point in time polynomial in  $\log \frac{1}{\varepsilon}$ . In the full paper, we give examples of problems with bounded width arising in combinatorial optimization.

### 3 Relation to Previous Work

**Matrix and Polyhedral Games.** The special case when each of the sets  $X$  and  $Y$  is a polytope (or more generally, a polyhedron) and payoff is a bilinear function, is known as polyhedral games (see e.g. [Was03]). When each of these polytopes is just a simplex we obtain the well-known class of matrix games. Even though each polyhedral game can be reduced to a matrix game by using the vertex representation of each polytope (see e.g. [Sch86]), this transformation

---

<sup>2</sup> See Section 5.3 for details.

<sup>3</sup> Here, we apply random sampling as a black box for each iteration independently; it might be possible to improve the running time if we utilize the fact that the distributions are only slightly modified from an iteration to the next.

<sup>4</sup>  $O^*(\cdot)$  suppresses polylogarithmic factors that depend on  $n$ ,  $m$  and  $\varepsilon$ .

may be (and is typically) not algorithmically efficient since the number of vertices may be exponential in the number of facets by which each polytope is given.

**Fictitious Play.** We assume for the purposes of this subsection that both sets  $X$  and  $Y$  are closed, and hence the infimum and supremum in (1) are replaced by the minimum and maximum, respectively.

In *fictitious play* which is a deterministic procedure and originally proposed by Brown [Bro51] for matrix games, each player updates his/her strategy by applying the best response, given the current opponent's strategy. More precisely, the minimizer and the maximizer initialize, respectively,  $x(0) = 0$  and  $y(0) = 0$ , and for  $t = 1, 2, \dots$ , update  $x(t)$  and  $y(t)$  by

$$x(t+1) = \frac{t}{t+1}x(t) + \frac{1}{t+1}\xi(t), \text{ where } \xi(t) = \operatorname{argmin}_{\xi \in X} F(\xi, y(t)), \quad (3)$$

$$y(t+1) = \frac{t}{t+1}y(t) + \frac{1}{t+1}\eta(t), \text{ where } \eta(t) = \operatorname{argmax}_{\eta \in Y} F(x(t), \eta). \quad (4)$$

For matrix games, the convergence to optimal strategies was established by Robinson [Rob51]. In this case, the best response of each player at each step can be chosen from the vertices of the corresponding simplex. A bound of  $\left(\frac{2^{m+n}}{\varepsilon}\right)^{m+n-2}$  on the time needed for convergence to an  $\varepsilon$ -saddle point was obtained by Shapiro [Sha58]. More recently, Hofbauer and Sorin [HS06] showed the convergence of fictitious play for continuous convex-concave functions over compact convex sets; they also gave a dynamic proof for the minmax theorem.

**Randomized Fictitious Play.** In [GK95], Grigoriadis and Khachiyan introduced a randomized variant of fictitious play for matrix games. Their algorithm replaces the minimum and maximum selections (3)-(4) by a smoothed version, in which, at each time step  $t$ , the minimizing player selects a strategy  $i \in [m]$  with probability proportional to  $\exp\{-\frac{\varepsilon}{2}e_i^T A y(t)\}$ , where  $e_i$  denotes the  $i$ th unit vector of dimension  $m$ . Similarly, the maximizing player chooses strategy  $j \in [n]$  with probability proportional to  $\exp\{\frac{\varepsilon}{2}x(t)^T A e_j\}$ . Grigoriadis and Khachiyan proved that, if  $A \in [-1, 1]^{m \times n}$ , then this algorithm converges, with high probability, to an  $\varepsilon$ -saddle point in  $O(\frac{\log(m+n)}{\varepsilon^2})$  iterations. Our result builds on [GK95].

**The Multiplicative Weights Update Method.** Freund and Schapire [FS99] showed how to use the *weighted majority algorithm*, originally developed by Littlestone and Warmuth [LW94], for computing  $\varepsilon$ -saddle points for matrix games. Their procedure can be thought of as a derandomization of the randomized fictitious play described above. Similar algorithms have also been developed for approximately solving special optimization problems, such as general linear programs [PST91], multicommodity flow problems [GK98], packing and covering linear programs [PST91, GK98, GK04, KY07, You01], a class of convex programs [Kha04], and semidefinite programs [AHK05, AK07]. Arora, Hazan and Kale [AHK06] consider the following scenario: given a finite set  $X$  of decisions and a finite set  $Y$  of outputs, and a payoff matrix  $M \in \mathbb{R}^{X \times Y}$  such that  $M(x, y)$  is the penalty that would be paid if decision  $x \in X$  was made and output  $y \in Y$

was the result, the objective is to develop a decision making strategy that tends to minimize the total payoff over many rounds of such decision making. Arora et al. [AHK06, Kal07] show how to apply this framework to approximately computing  $\max_{y \in Y} \min_{i \in [m]} f_i(y)$ , given an oracle for finding  $\max_{y \in Y} \sum_{i \in [m]} \lambda_i f_i(y)$  for any non-negative  $\lambda \in \mathbb{R}^m$  such that  $\sum_{i=1}^m \lambda_i = 1$ , where  $Y \subseteq \mathbb{R}^n$  is a given convex set and  $f_1, \dots, f_m : Y \rightarrow \mathbb{R}$  are given concave functions (see also [Kha04] for similar results).

There are two reasons why this method cannot be (directly) used to solve our problem (2). First, the number of decisions  $m$  is infinite in our case, and second, we do not assume to have access to an oracle of the type described above; we assume only a (weakest possible) membership oracle on  $Y$ . Our algorithm *extends* the multiplicative update method to the computation of approximate saddle points.

**Hazan's Work.** In his Ph.D. Thesis [Haz06], Hazan gave an algorithm, based on the multiplicative weights updates method, for approximating the minimum of a convex function within an absolute error of  $\varepsilon$ . Theorem 4.14 in [Haz06] suggests that a similar procedure<sup>5</sup> can be used to approximate a saddle point for convex-concave functions, however, without a running time analysis.

**Sampling Algorithms.** Our algorithm makes use of known algorithms for sampling from a given log-concave distribution<sup>6</sup>  $f(\cdot)$  over a convex set  $X \subseteq \mathbb{R}^m$ . The currently best known result achieving this is due to Lovász and Vempala (see, e.g., [LV07, Theorem 2.1]): a random walk on  $X$  converges in  $O^*(m^5)$  steps to a distribution within a total *variation distance* of  $\varepsilon$  from the desired exponential distribution with high probability.

Several algorithms for convex optimization based on sampling have been recently proposed. Bertsimas and Vempala [BV04] showed how to minimize a convex function over a convex set  $X \subseteq \mathbb{R}^m$ , given by a membership oracle, in time  $O^*((m^5 C + m^7) \log R)$ , where  $C$  is the time required by a single oracle call. When the function is linear time  $O^*(m^{4.5} C)$  suffices (Kalai and Vempala [KV06]).

Note that we can write (1) as the convex minimization problem  $\inf_{x \in X} F(x)$ , where  $F(x) = \sup_{y \in Y} F(x, y)$  is a convex function. Thus, it is worth comparing the bounds we obtain in Theorem 1 with the bounds that one could obtain by applying the random sampling techniques of [BV04, KV06] (see Table 1 in [BV04] for a comparison between these techniques and the Ellipsoid method). Since the above program is equivalent to  $\inf\{v : x \in X, \text{ and } F(x, y) \leq v \text{ for all } y \in Y\}$ , the solution can be obtained by applying the technique of [BV04, KV06], where each membership call involves another application of these techniques (to check if  $\sup_{y \in Y} F(x, y) \leq v$ ). The total time required by this procedure is

<sup>5</sup> This procedure can be written in the same form as our Algorithm 1 below, except that it chooses the points  $\xi(t) \in X$  and  $\eta(t) \in Y$ , at each time step  $t = 1, \dots, T$ , as the (approximate) centroids of the corresponding sets with respect to densities  $p_\xi(t) = e^{\sum_{\tau=1}^{t-1} \ln(e - F(\xi, \eta(\tau)))}$  and  $q_\eta(t) = e^{\sum_{\tau=1}^{t-1} \ln(e + F(\xi(\tau), \eta)))}$  (both of which are log-concave distributions), and outputs  $(\frac{1}{T} \sum_{t=1}^T \xi(t), \frac{1}{T} \sum_{t=1}^T \eta(t))$  at the end.

<sup>6</sup> That is,  $\log f(\cdot)$  is concave.

**Algorithm 1.** Randomized fictitious play

---

**Input:** Two convex bounded sets  $X, Y$  and a function  $F(x, y)$  such that  $F(\cdot, y) : X \rightarrow \mathbb{R}$  is convex for all  $y \in Y$  and  $F(x, \cdot) : Y \rightarrow \mathbb{R}$  is concave for all  $x \in X$ , satisfying assumptions (A1) and (A2)

**Output:** A pair of  $\varepsilon$ -optimal strategies

- 1:  $t := 0$ ; choose  $x(0) \in X$ ;  $y(0) \in Y$ , arbitrarily
  - 2: **while**  $t \leq T$  **do**
  - 3:   Pick  $\xi \in X$  and  $\eta \in Y$ , independently, from  $X$  and  $Y$  with densities  $\frac{p_\xi(t)}{\|p(t)\|_1}$  and  $\frac{q_\eta(t)}{\|q(t)\|_1}$ , respectively
  - 4:    $x(t+1) := \frac{t}{t+1}x(t) + \frac{1}{t+1}\xi$ ;  $y(t+1) := \frac{t}{t+1}y(t) + \frac{1}{t+1}\eta$ ;  $t := t + 1$ ;
  - 5: **end while**
  - 6: **return**  $(x(t), y(t))$
- 

$O^*(n^{4.5}(m^5C + m^7)\log R)$ , which is significantly greater than the bound stated in Theorem 1.

## 4 The Algorithm

The algorithm (see box 1) is a direct generalization of the algorithm in [GK95]. It proceeds in steps  $t = 0, 1, \dots$ , updating the pair of *accumulative* strategies  $x(t)$  and  $y(t)$ . Given the current pair  $(x(t), y(t))$ , define

$$p_\xi(t) = e^{-\frac{\varepsilon t F(\xi, y(t))}{2}} \quad \text{for } \xi \in X, \quad (5)$$

$$q_\eta(t) = e^{\frac{\varepsilon t F(x(t), \eta)}{2}} \quad \text{for } \eta \in Y, \quad (6)$$

and let

$$\|p(t)\|_1 = \int_{\xi \in X} p_\xi(t) d\xi \quad \text{and} \quad \|q(t)\|_1 = \int_{\eta \in Y} q_\eta(t) d\eta.$$

The parameter  $T$  will be specified later (see Lemma 4).

## 5 Analysis

Following [GK95], we use a potential function  $\Phi(t) = \|p(t)\|_1\|q(t)\|_1$  to bound the number of iterations required by the algorithm to reach an  $\varepsilon$ -saddle point. The analysis is composed of three parts. The first part of the analysis (Section 5.1), is a generalization of the arguments in [GK95] (and [KY07]): we show that the potential function increases, on the average, only by a factor of  $e^{O(\varepsilon^2)}$ , implying that after  $t$  iterations the potential is at most a factor of  $e^{O(\varepsilon^2)t}$  of the initial potential. While this was enough to bound the number of iterations by  $\varepsilon^{-2} \log(n+m)$  when both  $X$  and  $Y$  are simplices and the potential is a sum over all vertices of the simplices [GK95], this cannot be directly applied in our case.

This is because the fact that a definite integral of a non-negative function over a given region  $Q$  is bounded by some  $\tau$  does not imply that the function at any point in  $Q$  is also bounded by  $\tau$ . In the second part of the analysis (Section 5.2), we overcome this difficulty by showing that, due to concavity of the exponents in (5) and (6), the change in the function around a given point cannot be too large, and hence, the value at a given point cannot be large unless there is a sufficiently large fraction of the volume of the sets  $X$  and  $Y$  over which the integral is also too large. In the last part of the analysis (Section 5.3), we show that the same bound on the running time holds when the sampling distributions in line 3 of the algorithm are replaced by sufficiently close approximate distributions.

### 5.1 Bounding the Potential Increase

**Lemma 1.** *For  $t = 0, 1, 2, \dots$ ,*

$$\mathbb{E}[\Phi(t+1)] \leq \mathbb{E}[\Phi(t)]\left(1 + \frac{\varepsilon^2}{6}\right)^2.$$

The proof is a direct generalization of the proof in [GK95] and is given in the full paper. By Markov's inequality we have the following statement.

**Corollary 1.** *With probability at least  $\frac{1}{2}$ , after  $t$  iterations,*

$$\Phi(t) \leq 2e^{\frac{\varepsilon^2}{3}t}\Phi(0). \quad (7)$$

At this point one might tend to conclude the proof, as in [GK95, KY07], by implying from Corollary 1 and the non-negativity of the function under the integral

$$\Phi(t) = \int_{\xi \in X, \eta \in Y} e^{\frac{\varepsilon}{2}t(F(x(t), \eta) - F(\xi, y(t)))} d\xi d\eta, \quad (8)$$

that this function is bounded at every point also by  $2e^{\frac{\varepsilon^2}{3}t}\Phi(0)$  (with high probability). This would then imply that the current strategies are  $\varepsilon$ -optimal. However, this is not necessarily true in general and we have to modify the argument to show that, even though the value of the function at some points can be larger than the bound  $2e^{\frac{\varepsilon^2}{3}t}\Phi(0)$ , the increase in this value cannot be more than an exponential (in the input description), which would be still enough for the bound on the number of iterations to go through.

### 5.2 Bounding the Number of Iterations

For convenience, define  $Z = X \times Y$ , and concave function  $g_t : Z \rightarrow \mathbb{R}$  given at any point  $z = (\xi, \eta) \in Z$  by  $g_t(\xi, \eta) := \frac{\varepsilon}{2}t(F(x(t), \eta) - F(\xi, y(t)))$ . Note that, by our assumptions,  $Z$  is a full-dimensional bounded convex set in  $\mathbb{R}^N$  of volume  $\Phi(0) = \text{vol}(X) \cdot \text{vol}(Y)$ , where  $N = n+m$ . Furthermore, assumption (A2) implies that  $|g_t(z)| = |\frac{\varepsilon}{2}t(F(x(t), \eta) - F(\xi, y(t)))| \leq \varepsilon t$  for all  $z \in Z$ .

A sufficient condition for the convergence to an  $\varepsilon$ -approximate equilibrium is provided by the following lemma.

**Lemma 2.** Suppose that (7) holds and there exists an  $\alpha$  such that

$$0 < \alpha < 4\varepsilon t, \quad (9)$$

$$e^{\frac{1}{2}\alpha} \left( \frac{\alpha}{4\varepsilon t} \right)^N \text{vol}(Z) > 1. \quad (10)$$

Then

$$e^{g_t(z)} \leq 2e^{\frac{\varepsilon^2}{3}t+\alpha}\Phi(0) \text{ for all } z \in Z. \quad (11)$$

*Proof.* Assume otherwise, i.e., there is  $z^* \in Z$  with  $g_t(z^*) > \frac{\varepsilon^2}{3}t + \alpha + \ln(2\Phi(0))$ . Let  $\lambda^* = \alpha/(4\varepsilon t)$ ,

$$Z^+ = \{z \in Z | g_t(z) \geq g_t(z^*) - \alpha/2\}, \text{ and } Z^{++} = \{z^* + \frac{1}{\lambda^*}(z - z^*) | z \in Z^+\}.$$

Concavity of  $g_t$  implies convexity of  $Z^+$ . This implies in particular that  $z^* + \lambda'(z - z^*) \in Z^{++}$  of all  $0 \leq \lambda' \leq \frac{1}{\lambda^*}$  and  $z \in Z^+$ , since  $z^* + \lambda^*\lambda'(z - z^*) \in Z^+$ . We claim that  $Z \subseteq Z^{++}$ . Assume otherwise, and let  $x \in Z \setminus Z^{++}$  (and hence  $x \in Z \setminus Z^+$ ). Let

$$\lambda^+ = \sup\{\lambda | z^* + \lambda(x - z^*) \in Z^+\} \text{ and } z^+ = z^* + \lambda^+(x - z^*).$$

By continuity of  $g_t$ ,  $z^+ \in Z^+$  and  $g_t(z^*) - \alpha/2 = g_t(z^+)$ . We have  $x - z^* = \frac{1}{\lambda^+}(z^+ - z^*)$  and hence  $\frac{1}{\lambda^+} > \frac{1}{\lambda^*}$ . But  $z^+ = \lambda^+x + (1 - \lambda^+)z^*$  and hence

$$g_t(z^*) - \alpha/2 = g_t(z^+) = g_t(\lambda^+x + (1 - \lambda^+)z^*) \geq \lambda^+g_t(x) + (1 - \lambda^+)g_t(z^*).$$

Thus

$$\frac{\alpha}{2} \leq \lambda^+(g_t(z^*) - g_t(x)) \leq 2\varepsilon t \lambda^+,$$

a contradiction. We have now established  $Z \subseteq Z^{++}$ . The containment implies

$$\text{vol}(Z) \leq \text{vol}(Z^{++}) = \left( \frac{1}{\lambda^*} \right)^N \text{vol}(Z^+)$$

and further

$$\begin{aligned} \Phi(t) &= \int_{z \in Z} e^{g_t(z)} dz \geq \int_{z \in Z^+} e^{g_t(z)} dz \\ &\geq 2\Phi(0)e^{\frac{\varepsilon^2}{3}t+\frac{1}{2}\alpha} \text{vol}(Z^+) \geq 2\Phi(0)e^{\frac{\varepsilon^2}{3}t+\frac{1}{2}\alpha} \left( \frac{\alpha}{4\varepsilon t} \right)^N \text{vol}(Z) > 2\Phi(0)e^{\frac{\varepsilon^2}{3}t}, \end{aligned}$$

a contradiction to (7).  $\square$

We can now derive an upper-bound on the number of iterations needed to converge to  $\varepsilon$ -optimal strategies.

**Lemma 3.** If (11) holds and

$$t \geq \frac{6}{\varepsilon^2}(\alpha + \max\{0, \ln(2\text{vol}(Z))\}), \quad (12)$$

then  $(x(t), y(t))$  is an  $\varepsilon$ -optimal pair.

*Proof.* By (11) we have  $g_t(z) \leq \frac{\varepsilon^2}{3}t + \alpha + \ln(2\Phi(0)) = \frac{\varepsilon^2}{3}t + \alpha + \ln(2\text{vol}(Z))$  for all  $z \in Z$ , or equivalently,

$$\frac{\varepsilon}{2}t(F(x(t), \eta) - F(\xi, y(t))) \leq \frac{\varepsilon^2}{3}t + \alpha + \ln(2\text{vol}(Z)) \quad \text{for all } \xi \in X \text{ and } \eta \in Y.$$

Hence,

$$F(x(t), \eta) \leq F(\xi, y(t)) + \frac{2\varepsilon}{3} + \frac{2}{\varepsilon t}(\alpha + \ln(2\text{vol}(Z))) \quad \text{for all } \xi \in X \text{ and } \eta \in Y,$$

which implies by (12) that

$$F(x(t), \eta) \leq F(\xi, y(t)) + \varepsilon \quad \text{for all } \xi \in X \text{ and } \eta \in Y.$$

□

**Lemma 4.** *For any  $\varepsilon \in (0, 1)$ , there exist  $\alpha$  and*

$$t = O\left(\frac{N}{\varepsilon^2} \ln \frac{R}{\varepsilon}\right)$$

satisfying (9), (10) and (12).

*Proof.* Assume  $\text{vol}(Z) \leq \frac{1}{2}$ . Let us choose  $t = \frac{6\alpha}{\varepsilon^2}$ . Then (10) becomes (after taking logarithms)

$$\frac{\alpha}{2} + N \ln\left(\frac{\alpha}{4\varepsilon t}\right) + \ln(\text{vol}(Z)) > 0.$$

So choosing  $\frac{\alpha}{2} = N \ln\left(\frac{25}{\varepsilon}\right) - \ln(\text{vol}(Z))$  would satisfy this inequality. Then

$$t = O\left(\frac{N}{\varepsilon^2} \ln \frac{1}{\varepsilon} + \frac{1}{\varepsilon^2} \ln \frac{1}{\text{vol}(Z)}\right).$$

Since  $1/\text{vol } Z \leq R^N$ , the claim follows. Let now  $\text{vol}(Z) > \frac{1}{2}$ . Then

$$e^{\frac{\alpha}{2}}\left(\frac{\alpha}{4\varepsilon t}\right)^N \text{vol}(Z) > \frac{1}{2}e^{\frac{\alpha}{2}}\left(\frac{\alpha}{4\varepsilon t}\right)^N,$$

Thus, in order to satisfy (10), it is enough to find  $\alpha$  and  $t$  satisfying

$$\frac{1}{2}e^{\frac{\alpha}{2}}\left(\frac{\alpha}{4\varepsilon t}\right)^N > 1.$$

To satisfy (12), let us simply choose  $t = \frac{6\alpha}{\varepsilon^2} + \frac{6}{\varepsilon^2} \ln(2\text{vol}(Z))$  and demand that

$$\frac{1}{2}e^{\frac{\alpha}{2}}\left(\frac{\alpha}{4\varepsilon t}\right)^N = \frac{1}{2}e^{\frac{\alpha}{2}}\left(\frac{\alpha}{\frac{24\alpha}{\varepsilon} + \frac{24}{\varepsilon} \ln(2\text{vol}(Z))}\right)^N > 1,$$

or equivalently,

$$2\left(\frac{24}{\varepsilon}\right)^N \left(1 + \frac{\ln(2\text{vol}(Z))}{\alpha}\right)^N < e^{\frac{\alpha}{2}}.$$

Thus, it is enough to select  $\alpha = \max \left\{ 4(\ln 2 + N \ln(\frac{24}{\varepsilon})), 2\sqrt{N \ln(2 \text{vol}(Z))} \right\}$  which satisfies

$$2 \left( \frac{24}{\varepsilon} \right)^N \leq e^{\frac{\alpha}{4}} \quad \text{and} \quad \left( 1 + \frac{\ln(2 \text{vol}(Z))}{\alpha} \right)^N < e^{\frac{\ln(2 \text{vol}(Z))}{\alpha} N} \leq e^{\frac{\alpha}{4}}.$$

It follows that

$$t = \max \left\{ \frac{24}{\varepsilon^2} (\ln 2 + N \ln(\frac{24}{\varepsilon})), \frac{12}{\varepsilon^2} \sqrt{N \ln(2 \text{vol}(Z))} \right\} + \frac{6}{\varepsilon^2} \ln(2 \text{vol}(Z)).$$

Since  $\text{vol}(Z) \leq R^N$ , the claim follows.  $\square$

Setting T in Algorithm 1 to the value of t given by Lemma 4 yields the following result.

**Corollary 2.** *Assume  $X$  and  $Y$  satisfy assumptions (A1) and (A2). Then Algorithm 1 computes a pair of  $\varepsilon$ -optimal strategies in expected  $O(\frac{n+m}{\varepsilon^2} \ln \frac{R}{\varepsilon})$  iterations.*

### 5.3 Using Approximate Distributions

We now consider the (realistic) situation when we can only sample approximately from the convex sets. In this case we assume the existence of approximate sampling routines that, upon the call in step 3 of the algorithm, return vectors  $\xi \in X$ , and (independently)  $\eta \in Y$ , with densities  $\hat{p}_\xi(t)$  and  $\hat{q}_\eta(t)$ , such that

$$\sup_{X' \subseteq X} \left| \frac{\hat{p}_{X'}(t)}{\hat{p}_X(t)} - \frac{p_{X'}(t)}{p_X(t)} \right| \leq \delta \quad \text{and} \quad \sup_{Y' \subseteq Y} \left| \frac{\hat{q}_{Y'}(t)}{\hat{q}_Y(t)} - \frac{q_{Y'}(t)}{q_Y(t)} \right| \leq \delta, \quad (13)$$

where  $\hat{p}_{X'}(t) = \int_{\xi \in X'} \hat{p}_\xi d\xi$  (similarly, define  $p_{X'}(t), \hat{q}_{Y'}(t), q_{Y'}(t)$ ), and  $\delta$  is a given desired accuracy. We next prove an approximate version of Lemma 1.

**Lemma 5.** *Suppose that we use approximate sampling routines with  $\delta = \varepsilon/4$  in step 3 of Algorithm 1. Then, for  $t = 0, 1, 2, \dots$ , we have*

$$\mathbb{E}[\Phi(t+1)] \leq \mathbb{E}[\Phi(t)] \left( 1 + \frac{43}{36} \varepsilon^2 \right).$$

See the full paper for the proof. Combining the currently known bound on the mixing time for sampling (see [LV04, LV06, LV07] and also Section 3) with the bounds on the number of iterations from Corollary 2 gives Theorem 1.

## 6 Conclusion and Acknowledgment

We showed that randomized fictitious play can be applied for computing  $\varepsilon$ -saddle points of convex-concave functions over the product of two convex bounded sets. Even though our bounds were stated for general convex sets, one should note that these bounds may be improved for classes of convex sets for which faster sampling procedures could be developed. We believe that the method used in this paper could be useful for developing algorithms for computing approximate equilibria for other classes of games.

We are grateful to Endre Boros and Vladimir Gurvich for many valuable discussions.

## References

- [AHK05] Arora, S., Hazan, E., Kale, S.: Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In: FOCS, pp. 339–348 (2005)
- [AHK06] Arora, S., Hazan, E., Kale, S.: Multiplicative weights method: a meta-algorithm and its applications. Technical report, Princeton University, USA (2006)
- [AK07] Arora, S., Kale, S.: A combinatorial, primal-dual approach to semidefinite programs. In: STOC, pp. 227–236 (2007)
- [BBR04] Bartal, Y., Byers, J.W., Raz, D.: Fast, distributed approximation algorithms for positive linear programming with applications to flow control. SIAM J. Comput. 33(6), 1261–1279 (2004)
- [Bro51] Brown, G.W.: Iterative solution of games by fictitious play. In: Koopmans, T.C. (ed.) Activity Analysis of Production and Allocation, pp. 374–376 (1951)
- [BV04] Bertsimas, D., Vempala, S.: Solving convex programs by random walks. J. ACM 51(4), 540–556 (2004)
- [Dan63] Dantzig, G.B.: Linear Programming and extensions. Princeton University Press (1963)
- [DJ07] Diedrich, F., Jansen, K.: Faster and simpler approximation algorithms for mixed packing and covering problems. Theor. Comput. Sci. 377, 181–204 (2007)
- [FS99] Freund, Y., Schapire, R.E.: Adaptive game playing using multiplicative weights. Games and Economic Behavior 29(1-2), 79–103 (1999)
- [GK92] Grigoriadis, M.D., Khachiyan, L.G.: Approximate solution of matrix games in parallel. In: Advances in Optimization and Parallel Computing, pp. 129–136 (1992)
- [GK94] Grigoriadis, M.D., Khachiyan, L.G.: Fast approximation schemes for convex programs with many blocks and coupling constraints. SIAM J. Optim. 4, 86–107 (1994)
- [GK95] Grigoriadis, M.D., Khachiyan, L.G.: A sublinear-time randomized approximation algorithm for matrix games. Operations Research Letters 18(2), 53–58 (1995)
- [GK96] Grigoriadis, M.D., Khachiyan, L.G.: Coordination complexity of parallel price-directive decomposition. Math. Oper. Res. 21(2), 321–340 (1996)
- [GK98] Garg, N., Könemann, J.: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In: FOCS, pp. 300–309 (1998)
- [GK04] Garg, N., Khandekar, R.: Fractional covering with upper bounds on the variables: Solving IPs with negative entries. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 371–382. Springer, Heidelberg (2004)
- [GKPV01] Grigoriadis, M.D., Khachiyan, L.G., Porkolab, L., Villavicencio, J.: Approximate max-min resource sharing for structured concave optimization. SIAM Journal on Optimization 41, 1081–1091 (2001)
- [GLS93] Grötschel, M., Lovász, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization, 2nd edn. Algorithms and Combinatorics, vol. 2. Springer (1993)
- [Haz06] Hazan, E.: Efficient Algorithms for Online Convex Optimization and Their Application. PhD thesis, Princeton University, USA (2006)

- [HS06] Hofbauer, J., Sorin, S.: Best response dynamics for continuous zero-sum games. *Discrete and Continuous Dynamical Systems – Series B* 6(1), 215–224 (2006)
- [Jan04] Jansen, K.: Approximation algorithms for mixed fractional packing and covering problems. In: IFIP TCS, pp. 223–236 (2004)
- [Kal07] Kale, S.: Efficient Algorithms using the Multiplicative Weights Update Method. PhD thesis, Princeton University, USA (2007)
- [Kha04] Khandekar, R.: Lagrangian Relaxation based Algorithms for Convex Programming Problems. PhD thesis, Indian Institute of Technology, Delhi (2004)
- [KV06] Kalai, A., Vempala, S.: Simulated annealing for convex optimization. *Math. Oper. Res.* 31(2), 253–266 (2006)
- [KY07] Koufogiannakis, C., Young, N.E.: Beating simplex for fractional packing and covering linear programs. In: FOCS, pp. 494–504 (2007)
- [LN93] Luby, M., Nisan, N.: A parallel approximation algorithm for positive linear programming. In: STOC, pp. 448–457 (1993)
- [LV04] Lovász, L., Vempala, S.: Hit-and-run from a corner. In: STOC, pp. 310–314 (2004)
- [LV06] Lovász, L., Vempala, S.: Fast algorithms for logconcave functions: Sampling, rounding, integration and optimization. In: FOCS, pp. 57–68 (2006)
- [LV07] Lovász, L., Vempala, S.: The geometry of logconcave functions and sampling algorithms. *Random Struct. Algorithms* 30(3), 307–358 (2007)
- [LW94] Littlestone, N., Warmuth, M.K.: The weighted majority algorithm. *Inf. Comput.* 108(2), 212–261 (1994)
- [PST91] Plotkin, S.A., Shmoys, D.B., Tardos, É.: Fast approximation algorithms for fractional packing and covering problems. In: FOCS, pp. 495–504 (1991)
- [Rob51] Robinson, J.: An iterative method of solving a game. *The Annals of Mathematics* 54(2), 296–301 (1951)
- [Roc70] Rockafellar, R.T.: Convex Analysis. Princeton Mathematical Series. Princeton University Press (1970)
- [Sch86] Schrijver, A.: Theory of Linear and Integer Programming. Wiley, New York (1986)
- [Sha58] Shapiro, H.N.: Note on a computation method in the theory of games. *Communications on Pure and Applied Mathematics* 11(4), 587–593 (1958)
- [Vem05] Vempala, S.S.: Geometric random walks: A survey. *Combinatorial and Computational Geometry* 52, 573–612 (2005)
- [Was03] Washburn, A.R.: Two-Person Zero-Sum Games. INFORMS (2003)
- [You01] Young, N.E.: Sequential and parallel algorithms for mixed packing and covering. In: FOCS, pp. 538–546 (2001)

# A Fast Algorithm for Data Collection along a Fixed Track

Otfried Cheong<sup>1</sup>, Radwa El Shawi<sup>2,3</sup>, and Joachim Gudmundsson<sup>2,3</sup>

<sup>1</sup> Korea Advanced Institute of Science and Technology, Republic of Korea<sup>\*</sup>  
otfried@kaist.edu

<sup>2</sup> University of Sydney, Australia<sup>\*\*</sup>  
joachim.gudmundsson@sydney.edu.au

<sup>3</sup> NICTA\*\*\*, Sydney, Australia  
radwa.elshawi@nicta.com.au

**Abstract.** Recent research shows that significant energy saving can be achieved in wireless sensor networks (WSNs) with a mobile base station that collects data from sensor nodes via short-range communications. However, a major performance bottleneck of such WSNs is the significantly increased latency in data collection due to the low movement speed of mobile base stations. In this paper we study the problem of finding a data collection path for a mobile base station moving along a fixed track in a wireless sensor network to minimize the latency of data collection. The main contribution is an  $O(mn \log n)$  expected time algorithm, where  $n$  is the number of sensors in the networks and  $m$  is the complexity of the fixed track.

## 1 Introduction

Wireless sensor networks (WSNs) are a well established technology for many application areas. Their main aims are to monitor physical or environmental conditions and to cooperatively pass their data through the network to a main location. Realizing the full potential of wireless sensor networks poses research challenges ranging from hardware and architectural issues, to programming languages and operating systems for sensor networks, to security concerns, to algorithms for sensor network deployment [14].

WSNs usually consist of a large number of sensor nodes, which are battery-powered tiny devices. These devices perform three basic tasks: (i) sample a physical quantity from the surrounding environment, (ii) process the acquired data, and (iii) transfer them through wireless communications to a data collection point called sink node or base station [1,7]. The traditional WSN architectures

---

\* O.C. was supported in part by NRF grant 2011-0016434 and in part by NRF grant 2011-0030044 (SRC-GAIA), both funded by the government of Korea.

\*\* J.G. was funded by the Australian Research Council FT100100755.

\*\*\* NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

are based on the assumption that the network is dense, so that any two nodes can communicate with each other through multihop paths. As a consequence, in most cases the sensors are assumed to be static. However, recently mobility has been introduced to WSNs and it has been shown to have several advantages, such as, increased connectivity, lower cost, higher reliability and higher energy efficiency [2,10]. An overview of Wireless Sensor Networks with Mobile Elements (WSN-MEs) can be found in the comprehensive survey by Di Francesco et al. [7].

WSN-MEs have in general three main components [14]:

**Regular sensor nodes** are the sources of information. They perform sensing and may also forward or relay messages in the network.

**Sinks** (base stations) are the destinations of information. A network usually has very few sinks.

**Special support nodes** perform a specific task, such as acting as intermediate data collectors or mobile gateways.

In the setting considered in this paper we have one mobile sink that moves on a fixed track. The model was introduced by Xing et al. [16]. Although this is a very restricted model it simplifies the motion control of the mobile sink and it improves the system reliability and has therefore been adopted by several existing mobile sensor systems [3]. An example is when the sink only can move along fixed cables between trees [12]. The objective is to find a continuous path of length at most  $L$  along the track and a set of trees rooted on the path that connect all the sensor nodes, such that the total Euclidean length of the trees is minimized [16]. An example is shown in Fig. 1.

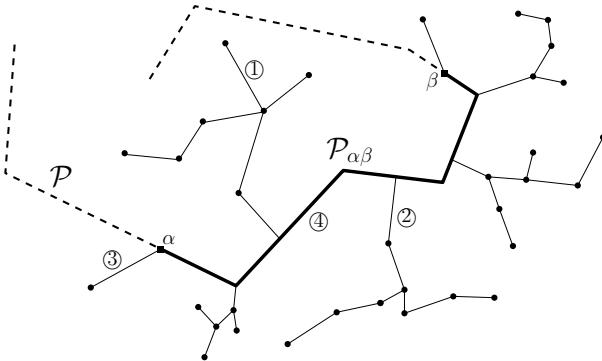
More formally, as input we are given a set  $\mathcal{S} = \{s_1, \dots, s_n\}$  of points (sensor nodes) together with a polygonal path  $\mathcal{P} = \langle p_1, \dots, p_m \rangle$  in  $\mathbb{R}^2$ . Given two points  $\alpha$  and  $\beta$  on  $\mathcal{P}$  (not necessarily vertices of  $\mathcal{P}$ ) let  $\mathcal{P}_{\alpha\beta}$  be the connected subpath of  $\mathcal{P}$  with  $\alpha$  and  $\beta$  as endpoints. Let  $MST(\mathcal{S}, \mathcal{P}_{\alpha\beta})$  be a minimum spanning tree of  $\mathcal{S}$  that contains  $\mathcal{P}_{\alpha\beta}$ , where an endpoint of an edge in  $MST(\mathcal{S}, \mathcal{P}_{\alpha\beta})$  can either be a point of  $\mathcal{S}$  or an arbitrary point on  $\mathcal{P}_{\alpha\beta}$ .

### *Problem 1. Data Collection on a Fixed Track (DCFT) problem*

Given a real value  $L$ , a set  $\mathcal{S} = \{s_1, \dots, s_n\}$  of points and a polygonal path  $\mathcal{P} = \langle p_1, \dots, p_m \rangle$  in  $\mathbb{R}^2$ , find a path  $\mathcal{P}_{\alpha\beta} \subseteq \mathcal{P}$  of length  $L$  such that  $wt(MST(\mathcal{S}, \mathcal{P}_{\alpha\beta}))$  is minimized, where  $wt(\cdot)$  denotes the total length of all the edges in the tree.

The path  $\mathcal{P}_{\alpha\beta}$  is called the *active* path. Since the weight of the active path is fixed the aim is to find a placement of the active path on  $\mathcal{P}$  that minimizes the total weight of the trees connecting  $\mathcal{S}$  to it. Note that a point in  $\mathcal{S}$  can be connected to any point along  $\mathcal{P}_{\alpha\beta}$ , not only to the vertices and endpoints of  $\mathcal{P}_{\alpha\beta}$ . We will sometimes write  $\mathcal{P}_{\alpha*}$  or  $\mathcal{P}_{*\beta}$  to denote the subpath of  $\mathcal{P}$  of length  $L$  starting at  $\alpha$  and ending at  $\beta$ , respectively.

To the best of the authors' knowledge very little work has been done on this problem from an algorithmic perspective and the only result we are aware of is the paper by Xing et al. [16]. They state without proof that the DCFT-problem is NP-hard, and concentrate on approximation algorithms. They showed how to



**Fig. 1.** Illustrating an instance of the DCFT problem with an active path  $\mathcal{P}_{\alpha\beta}$  and a minimum spanning tree connecting the sensor nodes to  $\mathcal{P}_{\alpha\beta}$ . The encircled numbers illustrate the edge types defined in Observation 1.

compute a  $3\varepsilon L$ -approximation for the DCFT-problem in  $O(\frac{wt(\mathcal{P})}{\varepsilon L} \cdot n \log n)$  time, where  $\varepsilon > 0$  is a given constant. In this paper we will show that the DCFT-problem can in fact be solved *exactly* in  $O(mn \log n)$  expected time (Theorem 5).

Omitted proofs can be found in the full version of the paper.

## 2 A Polynomial-Time Algorithm for the DCFT Problem

Since the length of  $\mathcal{P}_{\alpha\beta}$  is fixed, a natural approach to solve the problem is to sweep an active path of length  $L$  along  $\mathcal{P}$  while maintaining a minimum spanning tree. We will identify a set of  $O(mn)$  event points along  $\mathcal{P}$ . It will be shown that all topological changes to the minimum spanning tree that we maintain during the sweep will occur when the start or end point of the active path coincides with one of the event points.

Two problems need to be handled: (1) find all event points along  $\mathcal{P}$  of the sweep-line algorithm and (2) maintain a  $MST(\mathcal{S}, \mathcal{P}_{\alpha\beta})$  during the sweep.

### 2.1 Basic Properties and Notations

Given a point  $s$  and a segment  $\ell$  in the plane let  $op(s, \ell)$  denote the closest point on  $\ell$  to  $s$ , see Fig. 2(a). Similarly, let  $op(s, \mathcal{P}) = \cup_{\ell \in \mathcal{P}} op(s, \ell)$  and let  $op(\mathcal{S}, \mathcal{P}) = \cup_{s \in \mathcal{S}} op(s, \mathcal{P})$ .

Next we study the edges in an optimal solution in more detail (see Fig. 1).

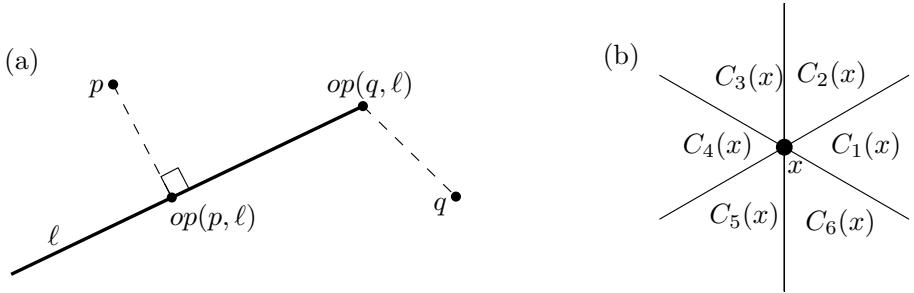
**Observation 1.** An edge  $(u, v) \in MST(\mathcal{S}, \mathcal{P}_{\alpha\beta})$  can be one of four types:

type-1:  $u, v \in \mathcal{S}$ ,

type-2:  $u \in \mathcal{S}$  and  $v \in op(u, \mathcal{P}_{\alpha\beta})$  and  $v$  lies on  $\mathcal{P}_{\alpha\beta}$ ,

type-3:  $u \in \mathcal{S}$  and  $v$  is either  $\alpha$  or  $\beta$ , and

type-4:  $u$  and  $v$  are consecutive vertices of  $\mathcal{P}$  or points in  $op(\mathcal{S}, \mathcal{P})$  along  $\mathcal{P}$  (these are the edges forming  $\mathcal{P}_{\alpha\beta}$ ).



**Fig. 2.** (a) Illustrating  $op(p, \ell)$  and  $op(q, \ell)$ . (b) Illustrating the definitions of  $C_i(x)$ ,  $1 \leq i \leq 6$ .

We will frequently refer to these edge types throughout this paper, not only referring to the edges of the spanning tree but to edges of any network.

Now we are ready to start constructing the set of event points, denoted  $\Gamma$ , along  $\mathcal{P}$ . The set will be the union of three subsets  $\Gamma_1$  (below),  $\Gamma_2$  (Theorem 2) and  $\Gamma_3$  (Theorem 4). Lets start with  $\Gamma_1$  which will contain the vertices of  $\mathcal{P}$  and the set of points  $op(\mathcal{S}, \mathcal{P})$ .

**Observation 2.** *The number of points in  $\Gamma_1$  is  $O(nm)$  and can be constructed in  $O(nm)$  time.*

We say that an active path  $\mathcal{P}'$  is swepted between two consecutive event points  $x$  and  $y$  in  $\Gamma_1$  if  $\mathcal{P}'$  starts with one endpoint coinciding with  $x$  and is then moved along  $\mathcal{P}$  until one of the end points of  $\mathcal{P}'$  coincides with  $y$ . During the sweep no other event point in  $\Gamma_1$  is encountered by any of the endpoints.

**Observation 3.** *Consider the inter-point Euclidean distance between two points  $u$  and  $v$  while sweeping the active path in-between two consecutive event points in  $\Gamma_1$ . If  $(u, v)$  is of type-1, type-2 or type-4 then the Euclidean distance  $|uv|$  is fixed during the sweep. If  $(u, v)$  is of type-3 then the Euclidean distance will monotonically increase or decrease.*

As a direct consequence we have that all events that may induce a topological change to a minimum spanning tree during a sweep between two consecutive event points in  $\Gamma_1$  will involve a type-3 edge. There are two cases:

**Case 1:** a type-3 edge will be replaced by another type-3 edge (Section 2.2), or  
**Case 2:** a type-3 edge will replace, or be replaced by, a type-1 or type-2 edge (Section 2.3).

## 2.2 Generating Event Points for Case 1

In the rest of this section we will only consider sweeping the active path in-between two event points in  $\Gamma_1$ . Consider an arbitrary point  $x$ . Let  $C_i(x)$ ,

$1 \leq i \leq 6$  be the six cones, ordered counter clockwise, that partition the plane into six cones with apex at  $x$  and interior angle  $\pi/3$ , see Fig. 2(b).

In the proof of Lemma 1 in [16], Xing et al. show the following observation (adapted to our notations).

**Observation 4.** *Let  $\mathcal{P}_{\alpha\beta}$  be an optimal solution. Assume  $\alpha$  lies on a segment  $(p_j, p_{j+1})$  of  $\mathcal{P}$  and assume  $(p_j, p_{j+1})$  is horizontal with  $p_j$  to the left of  $p_{j+1}$ . There exists a  $MST(\mathcal{S}, \mathcal{P}_{\alpha\beta})$  such that:*

- $\alpha$  is connected to at most one point in each  $C_i(\alpha)$ ,  $1 \leq i \leq 6$ , and  $\beta$  is connected to at most one point in each  $C_i(\beta)$ ,  $1 \leq i \leq 6$ , and
- no point of  $\mathcal{S}$  to the right of  $\alpha$ , is connected by an edge to  $\alpha$ . The symmetric result holds for  $\beta$ .

From the above observation it immediately follows that for each  $\alpha$  and for each  $\beta$  there are at most four *potential* type-3 edges in the minimum spanning tree. Thus, for a type-3 edge to be connected to an endpoint of the active path, say  $\alpha$ , in the  $MST(\mathcal{S}, \mathcal{P}_{\alpha\beta})$  it has to be a nearest neighbor to  $\alpha$  in one of the six cones  $C_i(\alpha)$ ,  $1 \leq i \leq 6$ .

Consider a case 1 event, and assume that a type-3 edge  $(s_1, \alpha)$  is replaced by another type-3 edge, say the type-3 edge  $(s_2, \gamma)$  where  $\gamma \in \{\alpha, \beta\}$ . Before the event point we have  $|s_1\alpha| < |s_2\gamma|$  and after the event point the order has changed to  $|s_1\alpha| > |s_2\gamma|$ .

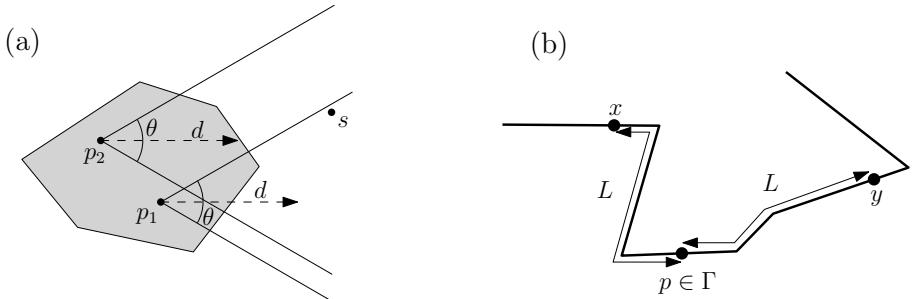
From the above discussion it follows that all case 1 events can be found if we, during the entire sweep, can keep track of the twelve (only eight of which are of interest) potential type-3 edges, i.e., the nearest neighbor in each  $C_i(\alpha)$  and  $C_i(\beta)$ ,  $1 \leq i \leq 6$ . We will use the following theorem that will be shown in Section 3. See Fig. 3(a) for an illustration.

**Theorem 1.** *Given a set  $\mathcal{S}$  of  $n$  points in the plane, a direction  $d$  and an angle  $\theta < \pi$  the plane can be partitioned into  $O(n)$  regions of total complexity  $O(n)$  in  $O(n \log n)$  expected time such that every point  $p$  in a region has the same nearest neighbor, say  $s$ , in  $C(p, d, \theta)$ , where  $C(p, d, \theta)$  is the cone with apex at  $p$ , interior angle  $\theta \leq \pi$  and with  $d$  as its bisector. The resulting partition is called an angle-restricted Voronoi diagram (ar-VD) of  $\mathcal{S}$ .*

Let  $\theta = \pi/3$  and consider the following six directions (counterclockwise angle formed with the positive  $x$ -axis)  $d_i = (i-1) \cdot \frac{\pi}{3}$ ,  $1 \leq i \leq 6$ , see Fig. 2(b). For each  $i$ ,  $1 \leq i \leq 6$ , construct an angle-restricted VD, denoted  $V_i$ , of  $\mathcal{S}$  with parameters  $\theta$  and  $d_i$  using Theorem 1.

Using the ar-VDs one can now construct the set  $\Gamma_2$ . Every intersection point between  $\mathcal{P}$  and an edge in any of the six ar-VDs  $V_1, \dots, V_6$ , is added to  $\Gamma_2$ . For each segment there are  $O(n)$  intersections, thus  $O(mn)$  in total. Each intersection point can be found in  $O(\log n)$  time using a standard point location data structure [5]. Constructing the six ar-VDs requires  $O(mn \log n)$  expected time according to Theorem 1, thus  $O(mn \log n)$  expected time in total.

Consider the sweep of the active path in-between two consecutive event points in  $\Gamma_1 \cup \Gamma_2$ . Note that during this sweep the potential type-3 edges will not



**Fig. 3.** (a) Every point  $p$  in a region of the angle-restricted Voronoi diagram have the same nearest neighbor in the cone  $C(p, d, \theta)$ . (b) Adding three vertices  $x, p$  and  $y$  to  $\mathcal{E}(\Lambda)$  along  $\mathcal{P}$  for each point in  $\Lambda$ .

change. Thus, to complete the set of all case 1 event points compute all events where the weight of two potential type-3 edges swap order. Since the weight of a potential type-3 edge is monotonically increasing or decreasing, according to Observation 3, there are at most 16 such events in-between two consecutive event points in  $\Gamma_1 \cup \Gamma_2$ . Given all the potential type-3 edges (at most eight) the events can be computed in constant time. Every point along  $\mathcal{P}$  where such an event takes place is added to  $\Gamma_2$ .

We summarize this section with the following theorem, which follows immediately from the above discussion.

**Theorem 2.** *All event points  $\Gamma_2$  where a type-3 edge is potentially replaced by another type-3 edge can be found in  $O(mn \log n)$  expected time using  $O(nm)$  space.*

### 2.3 Generating Event Points for Case 2

The final set of events is when a type-3 edge is replacing, or is replaced by, a type-1 or type-2 edge. We will show how the dynamic offline graph minimum spanning tree (DMST) algorithm by Eppstein [6] can be used to find  $\Gamma_3$ .

**Theorem 3.** *[Adapted from Theorem 1 in [6]] Given a sequence of  $k$  edge weight modifications in a graph of size  $N$ , starting from a state in which all weights are equal, we can compute the corresponding sequence of minimum spanning trees in time<sup>1</sup>  $O(k \log N)$  and space  $O(N)$ .*

**2.2.1 Build the Graph.** To use the above result we will construct a graph  $G = (V, E)$  and a sequence of edge weight modifications.

<sup>1</sup> In [6] the linear-time MST algorithm by Fredman and Willard [8] is used, resulting in  $O(k \log N)$  running time. If we are constrained to the real RAM model of computation we have to pay an additional factor of  $O(\log N)$  to build the MST, and thus an extra  $O(\log N)$  factor to the total running time.

Before we define the graph we need the following definition. Let  $\Lambda$  be a set of points along  $\mathcal{P}$ . From  $\Lambda$  one can generate the set  $\mathcal{E}(\Lambda)$  by adding up to three points on  $\mathcal{P}$  for each point  $p$  in  $\Lambda$ ; the point  $p$ , the two points at distance  $L$  along  $\mathcal{P}$  from  $p$ , if they exist. See Fig. 3(b) for an illustration.

The node set  $V$  corresponds to the points in  $\mathcal{S}$  and the points in  $\mathcal{E}(\Gamma_1 \cup \Gamma_2)$ , and its total size is  $O(mn)$ . The edge set  $E$  is constructed as follows:

1. Consider a minimum spanning tree of  $\mathcal{S}$ . Add the corresponding edges to  $E$ .
2. For every vertex  $v \in V$  corresponding to a point  $s \in \mathcal{S}$  add the  $O(m)$  edges connecting  $v$  to the set of vertices in  $V$  corresponding to the points  $op(s, \mathcal{P})$ .
3. For each segment  $e_i = (p_i, p_{i+1})$  in  $\mathcal{P}$ ,  $1 \leq i \leq m - 1$ , add an edge between the vertex in  $V$  corresponding to  $p_i$  and the vertex in  $V$  corresponding to the nearest neighbor in  $C_j(p_i)$ ,  $1 \leq j \leq 6$ .
4. For each point  $p \in op(\mathcal{S}, \mathcal{P})$  add an edge between the vertex in  $V$  corresponding to  $p$  and the vertex in  $V$  corresponding to the nearest neighbor in  $C_j(p)$ ,  $1 \leq j \leq 6$ .
5. For every two consecutive event points  $e_1$  and  $e_2$  in  $\mathcal{E}(\Gamma_1 \cup \Gamma_2)$  add an edge between the vertices in  $V$  corresponding to  $e_1$  and  $e_2$  to  $E$ .

Note that the number of edges added to  $E$  is  $O(mn)$ . Consider the time required for the five steps. Step 1 uses  $O(n \log n)$  time, steps 2 and 5 requires  $O(mn)$  time while steps 3 and 4 can be computed in  $O(mn \log n)$  time, using a standard point location query data structure on the ar-VDs.

**Lemma 1.** *Let  $\mathcal{P}_{\alpha\beta}$  be an active path with  $\alpha$  and/or  $\beta$  in  $\Gamma_1 \cup \Gamma_2$ . There exists a  $MST(\mathcal{S}, \mathcal{P}_{\alpha\beta})$  whose edge set is a subset of  $E$ .*

**2.2.2 Sweep the Active Path.** To be able to use Theorem 3 we initially set all edges in  $E$  to have unit weight. Next we build the sequence of edge weight modifications in two steps; one to initialize the sweep and one to simulate the sweep of the active path along  $\mathcal{P}$ .

*Initialization.* The active path starts at  $p_1$ , that is, the initial configuration is  $\mathcal{P}_{p_1*}$ . Next we modify the edge weights to simulate the original DCFT-problem. The weight of every type-1 edge  $(u, v)$  is set to  $|uv|$  and the weight of every type-4 edge is set to 0. The weight of a type-2 edge  $(u, v)$  is set to  $|uv|$  if  $v$  lies on the active path  $\mathcal{P}_{p_1*}$ , otherwise it is set to  $\infty$ . Similarly, the weight of a type-3 edge  $(u \in \mathcal{S}, v)$  is set to  $|uv|$  if  $v$  is one of the endpoints of  $\mathcal{P}_{p_1*}$ , otherwise it is set to  $\infty$ . Note that  $p_1 \in \Gamma_1$  and the opposite endpoint of  $\mathcal{P}_{p_1*}$  is in  $\mathcal{E}(\Gamma_1)$  thus, both are vertices in  $V$ . The total number of edge weight modifications needed for the initialization is  $O(mn)$ .

This correctly models the DCFT-problem for a fixed active path  $\mathcal{P}_{p_1*}$  and will include the edges in a  $MST(\mathcal{S}, \mathcal{P}_{p_1*})$ .

*Model the Sweep.* Next we compute the set of edge weight modifications to simulate sweeping the active path along  $\mathcal{P}$ . Assume that we have the active

path  $\mathcal{P}_{\alpha_1\beta_1}$  and that the algorithm correctly computed  $MST(\mathcal{S}, \mathcal{P}_{\alpha_1\beta_1})$ . Next we move the active path along  $\mathcal{P}$  until it encounters the next event point in  $\Gamma_1 \cup \Gamma_2$ , the new active path is denoted  $\mathcal{P}_{\alpha_2\beta_2}$ . The following edge weight modifications are performed:

- the weight of any type-2 edge connecting  $\alpha_1$  with a point in  $\mathcal{S}$  is set to  $\infty$ ,
- the weight of any type-2 edge connecting  $\beta_2$  with a point  $v$  in  $\mathcal{S}$  is set to  $|\beta_2 v|$ ,
- the weight of the potential type-3 edges connecting  $\alpha_1$  or  $\beta_1$  with points in  $\mathcal{S}$  is set to  $\infty$ , and
- the weight of the potential type-3 edges connecting  $\alpha_2$  or  $\beta_2$  with points in  $\mathcal{S}$  is set to be equal to the Euclidean distance between their endpoints,

This correctly models the DCFT-problem for the fixed active path  $\mathcal{P}_{\alpha_2\beta_2}$  and will include the edges in a  $MST(\mathcal{S}, \mathcal{P}_{\alpha_2\beta_2})$ . The total number of edge weight modifications is  $O(nm)$ , and each one can be computed in constant time provided that the six ar-VDs have been computed in a preprocessing step.

Given the initial graph  $G = (V, E)$  and the sequence of  $O(nm)$  edge weight modifications we can run Eppstein's algorithm (Theorem 3) in  $O(mn \log n)$  time. As output we obtain the corresponding sequence of minimum spanning trees (one for each event point), denoted  $T_1, \dots, T_k$ , where  $k = O(mn)$ .

**2.2.3 Compute the Event Points.** Recall that the aim of this section is to construct the set of points,  $\Gamma_3$ , along  $\mathcal{P}$  where a type-3 edge replaces, or is replaced by, a type-1 or type-2 edge. Above we only computed the minimum spanning trees for all the existing event points in  $\Gamma_1 \cup \Gamma_2$ , that is, for all cases when one of the endpoints of an active path coincides with a point in  $\Gamma_1 \cup \Gamma_2$ .

**Lemma 2.** *While sweeping the active path between two consecutive event points in  $\Gamma_1 \cup \Gamma_2$  the minimum spanning tree can change topology at most eight times.*

**Lemma 3.** *While sweeping the active path between two consecutive event points in  $\Gamma_1 \cup \Gamma_2$ , every point along the sweep where a topological change is made to the minimum spanning tree can be computed in  $O(\log n)$  time.*

Between every two consecutive event points in  $\Gamma_1 \cup \Gamma_2$  compute all the event points where a topological change is made to the minimum spanning tree, according to Lemma 3. These points form the set  $\Gamma_3$ .

The following theorem concludes this section:

**Theorem 4.** *All event points,  $\Gamma_3$ , when a type-3 edge is potentially replacing, or is potentially replaced by, a type-1 or type-2 edge can be found in  $O(mn \log n)$  expected time using  $O(nm)$  space.*

## 2.4 Maintaining the Spanning Tree

We can now merge the results. Let  $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3$ . Consider the sweep-line algorithm introduced in the first paragraph of Section 2. That is, sweep an active

path  $\mathcal{P}_{\alpha\beta}$  of length  $L$  along  $\mathcal{P}$  starting with  $\alpha = p_1$  and ending with  $\beta = p_m$ . During the sweep maintain a minimum spanning tree of  $\mathcal{S}$  and  $\mathcal{P}_{\alpha\beta}$ .

From Observation 3, Lemma 2 and Theorem 4 it follows that  $\Gamma$  contains all the event points where a minimum spanning tree might change its topology. Next simulate a sweep while maintaining the tree. We can use the same approach as we used in Section 2.2.2, but using  $\mathcal{S}$  and  $\mathcal{E}(\Gamma)$  as the set of vertices instead of  $\mathcal{S}$  and  $\mathcal{E}(\Gamma_1 \cup \Gamma_2)$ . Again, it is easy to see that this correctly models the DCFT-problem for the fixed active path  $\mathcal{P}_{\alpha\beta}$  and will include the edges in a  $MST(\mathcal{S}, \mathcal{P}_{\alpha\beta})$ . The total number of edge weight modifications is  $O(nm)$ , and each one can be computed in constant time provided that the six ar-VDs are computed in a preprocessing step.

Eppstein's algorithm generates the initial minimum spanning tree and the sequence of  $O(nm)$  changes made during the sweep. Consequently the weight of these trees can also be output without increasing the running time. Consider the sweep in-between two consecutive events in  $\Gamma$ . No topological changes are made to the minimum spanning tree. The only change is the weight of the type-3 edge. So to find a minimum weight spanning tree in-between two event points it is sufficient to compute the minimum of a function described by at most six distance functions, each distance function describing the minimum distance between a point and a straight-line segment. The minimum of this function can be computed in constant time, thus for each pair of consecutive event points,  $e_i$  and  $e_{i+1}$ , in  $\Gamma$  we can compute the minimum solution in-between  $e_i$  and  $e_{i+1}$  in constant time, given  $T_i$  and  $T_{i+1}$ . We can now summarize Section 2.

**Theorem 5.** *Given a real value  $L$ , a set  $\mathcal{S} = \{s_1, \dots, s_n\}$  of points and a polygonal path  $\mathcal{P} = \langle p_1, \dots, p_m \rangle$  in  $\mathbb{R}^2$ , an optimal solution for the DCFT-problem can be computed in  $O(mn \log n)$  expected time.*

### 3 Angle-Restricted Voronoi Diagrams

The abstract Voronoi diagram (AVD) [11] is used to construct the set of event points involving type-3 edges. In the following we show how the AVD is constructed and investigate some of its topological properties.

Chew and Drysdale's [4] divide-and-conquer algorithm for computing the Voronoi diagram under convex distance functions can be extended to the construction of angle-restricted Voronoi diagrams (ar-VD) with some modifications [15]. They proved that the ar-VD can be computed in  $O(n \log n)$  time but did not bound the complexity of the diagram (to the best of our knowledge), which we need to bound the size of  $\Gamma_3$ .

#### 3.1 Definition and Properties

A unifying approach to Voronoi diagrams was proposed by Klein [11], which is based on the concept of bisecting curves instead of distance functions. For any two sites,  $p$  and  $q$ , in  $S$ , let  $J(p, q)$  denote the curve that is homeomorphic

to a line and divides the plane into two open (unbounded) regions  $D(p, q)$  and  $D(q, p)$ , where  $D(p, q)$  contains  $p$  and  $D(q, p)$  contains  $q$ . The Voronoi region of  $p$  with respect to  $\mathcal{S}$ , denoted  $VR(p, \mathcal{S})$ , is the intersection of all  $D(p, q)$  regions as  $q$  varies in  $S \setminus \{p\}$ . The abstract Voronoi diagram is defined as:

$$AVD(p, \mathcal{S}) = \bigcup_{p \in \mathcal{S}} \partial(VR(p, \mathcal{S})).$$

Klein [11] showed that if the AVD is a “dominance system” (Definition 1) and the dominance system is “admissible” (Definition 2) then the AVD has many of the same properties as the concrete Voronoi diagrams. In particular, the complexity of the AVD is then  $O(n)$ , where  $n$  is the number of sites.

Our aim is to define a set of bisecting curves that fits the framework in [11]. First it has to be a dominance system.

**Definition 1.** *The family  $\mathcal{D} = \{D(p, q), p \neq q\}$  is called a dominance system over  $\mathcal{S}$  if the following holds:*

- $D(p, q)$  is a non-empty open subset of the plane,
- $J(p, q) = \cap(D(p, q))$  is homeomorphic to the open interval  $(0, 1)$ , and
- $\partial(D(p, q)) = \partial(D(q, p))$ , where  $\partial$  denotes the perimeter of a region.

We need to define the set  $\mathcal{J}$  of bisecting curves that divides the plane into two open (unbounded) regions  $D(p, q)$  and  $D(q, p)$  for each pair  $p, q \in \mathcal{S}$  of points, such that the resulting family  $\mathcal{D} = \{D(p, q), p \neq q\}$  is a dominance system. In Lemma 4 it will be shown that the abstract Voronoi diagram defined by these bisectors is a dominance system.

### 3.2 Defining Bisecting Curves

Given two points  $p$  and  $q$  in the plane let  $\mathcal{H}_p(p, q)$  be the set of points (the halfplane) whose Euclidean distance is smaller to  $p$  than to  $q$ . Furthermore, given a direction  $d$  and a point  $s$  in the plane let  $\ell_d(s)$  denote the infinite ray originating at  $s$  with direction  $d$ . Let  $C(s, d, \theta)$  be the cone with apex at  $s$ , angle  $\theta \leq \pi$  and with  $\ell_d(s)$  as its bisector.

For any two points  $p, q \in \mathcal{S}$  a bisecting curve  $J(p, q)$  is defined as follows. Assume w.l.o.g. that  $p$  lies to the left of  $q$  along the direction  $d$ . The bisecting curve  $J(p, q)$  is defined as the boundary of  $\{\mathcal{H}_q(p, q) \cap C(q, d, \theta)\}$ .

Given a direction  $d$ , let  $o_d$  denote the point at  $-\infty$  along the direction  $d$  and  $-\infty$  in the direction orthogonal to  $d$ .

**Lemma 4.** *Given a set  $\mathcal{S}$  of  $n$  points in the plane, a direction  $d$  and an angle  $\theta \leq \pi$  the family  $\mathcal{D} = \{D(p, q), p \neq q\}$  is a dominance system over  $\mathcal{S} \cup \{o_d\}$ .*

*Proof.* Consider the three properties in Definition 1. Adding  $o_d$  to our point set guarantees that the first property of the dominance system holds. The second and third properties follows immediately from the definition of bisectors.  $\square$

### 3.3 Computing the Diagram

The AVD defined by  $\mathcal{D}$  is denoted an *angle-restricted Voronoi diagram*. We now address the construction of the ar-VD( $\mathcal{S}$ ). Klein [11] showed that if a dominance system is *admissible* then the abstract Voronoi diagram has total size  $O(n)$ .

**Definition 2.** A dominance system  $\mathcal{J} = \{J(p, q) : p, q \in \mathcal{S}, p \neq q\}$  is called *admissible* if and only if for each subset  $\mathcal{S}'$  of  $\mathcal{S}$  of size at least three the following conditions are fulfilled:

- (a) The intersection of two bisecting curves consists of finitely many components.
- (b) The Voronoi regions are path-connected.
- (c) Each point of the plane lies in a Voronoi region or on the Voronoi diagram, i.e.  $\mathbb{R} = \bigcup_{p \in \mathcal{S}} VR(p)$ .

**Lemma 5.** The dominance system  $\mathcal{J} = \{J(p, q) : p, q \in S \cup o_d, p \neq q\}$  defined in Section 3.2 is admissible.

Mehlhorn et al. [9] and Klein et al. [13] have shown that one can, without further assumptions, apply the randomized incremental construction technique to abstract Voronoi diagrams.

**Theorem 6.** The abstract Voronoi diagram of an admissible system  $\mathcal{J} = \{J(p, q) : p, q \in \mathcal{S}, p \neq q\}$ , where  $|\mathcal{S}| = n$ , can be constructed within expected  $O(n \log n)$  many steps and expected space  $O(n)$ , by randomized incremental construction.

**Theorem 7.** Given a  $\mathcal{S}$  of points and a point  $q$  in the plane. The Voronoi region of the directed AVD( $S, d, \theta$ ) containing  $q$  corresponds to the point in  $\mathcal{S}$  closest to  $q$  in  $C(q, -d, \theta)$ .

*Proof.* The theorem follows immediately from the construction.  $\square$

Theorem 1 summarizes the results of this section and for convenience we restate it here.

**Theorem 1.** Given a set  $\mathcal{S}$  of  $n$  points in the plane, a direction  $d$  and an angle  $\theta < \pi$  the plane can be partitioned into  $O(n)$  regions of total complexity  $O(n)$  in  $O(n \log n)$  expected time such that every point  $p$  in a region has the same nearest neighbor, say  $s$ , in  $C(p, d, \theta)$ , where  $C(p, d, \theta)$  is the cone with apex at  $p$ , interior angle  $\theta \leq \pi$  and with  $d$  as its bisector. The resulting partition is called an angle-restricted Voronoi diagram (ar-VD) of  $\mathcal{S}$ .

## References

1. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. Computer Networks 38(4), 393–422 (2002)

2. Anastasi, G., Conti, M., Di Francesco, M., Passarella, A.: Energy conservation in wireless sensor networks: a survey. *Ad Hoc Networks* 7(3), 537–568 (2009)
3. Batalin, M., Rahimi, M., Yu, Y., Liu, D., Kansal, A., Sukhatme, G., Kaiser, W.J., Hansen, M., Pottie, G.J., Srivastava, M.B., Estrin, D.: Call and response: experiments in sampling the environment. In: Proceedings of the 2nd International ACM Conference on Embedded Networked Sensor Systems (Sensys), pp. 25–38 (2004)
4. Chew, L.P., Drysdale, R.L.: Voronoi diagrams based on convex distance functions. In: Proceedings of the 1st Annual Symposium on Computational Geometry (SoCG), pp. 235–244 (1985)
5. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational geometry – algorithms and applications, 3rd edn. Springer, Heidelberg (2008)
6. Eppstein, D.: Offline algorithms for dynamic minimum spanning tree problems. *Journal of Algorithms* 17(2), 237–250 (1994)
7. Di Francesco, M., Das, S.K., Anastasi, G.: Data collection in wireless sensor networks with mobile elements: a survey. *ACM Transactions on Sensor Networks* 8(1), 1–7 (2011)
8. Fredman, M.L., Willard, D.E.: Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Journal of Computer and System Sciences* 48(3), 533–551 (1994)
9. Meiser, S., Mehlhorn, K., O'Dúnlaing, C.: On the construction of abstract Voronoi diagrams. *Discrete and Computational Geometry* 6, 211–224 (1991)
10. Kansal, A., Somasundara, A., Jea, D., Srivastava, M., Estrin, D.: Intelligent fluid infrastructure for embedded networks. In: Proceedings of the 2nd ACM International Conference on Mobile Systems, Applications, and Services (MobiSys), pp. 111–124 (2004)
11. Klein, R.: Concrete and Abstract Voronoi Diagrams. LNCS, vol. 400. Springer, Heidelberg (1989)
12. Pon, R., Batalin, M.A., Gordon, J., Kansal, A., Liu, D., Rahimi, M., Shirachi, L., Yu, Y., Hansen, M., Kaiser, W.J., Srivastava, M., Sukhatme, G., Estrin, D.: Networked infomechanical systems: a mobile embedded networked sensor platform. In: Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN), pp. 376–381 (2005)
13. Mehlhorn, K., Klein, R., Meiser, S.: Randomized incremental construction of abstract Voronoi diagrams. *Computational Geometry: Theory and Applications* 3, 157–184 (1993)
14. Sahni, S., Xu, X.: Algorithms for wireless sensor networks. *International Journal of Distributed Sensor Networks* 1(1), 35–56 (2005)
15. Wee, Y.C., Chaiken, S., Willard, D.E.: General metrics and angle restricted Voronoi diagrams. In: Proceedings of the 1st Canadian Conference on Computational Geometry (1989)
16. Xing, G., Wang, T., Jia, W., Li, M.: Rendezvous design algorithms for wireless sensor networks with a mobile base station. In: Proceedings of the 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing, pp. 231–240 (2008)

# Random Methods for Parameterized Problems\*

Qilong Feng<sup>1</sup>, Jianxin Wang<sup>1</sup>, Shaohua Li<sup>1</sup>, and Jianer Chen<sup>1,2</sup>

<sup>1</sup> School of Information Science and Engineering,  
Central South University,  
Changsha 410083, P.R. China

<sup>2</sup> Department of Computer Science and Engineering  
Texas A&M University  
College Station, Texas 77843-3112, USA

**Abstract.** In this paper, we study the random methods for parameterized problems. For the Parameterized  $P_2$ -Packing problem, by randomly partitioning the vertices, a randomized parameterized algorithm of running time  $O^*(6.75^k)$  is obtained, improving the current best result  $O^*(8^k)$ . For the Parameterized Co-Path Packing problem, we study the kernel and randomized algorithm for the degree-bounded instance, and then by using the iterative compression technique, a randomized algorithm of running time  $O^*(3^k)$  is given for the Parameterized Co-Path Packing problem, improving the current best result  $O^*(3.24^k)$ .

## 1 Introduction

Random techniques have been used widely in designing parameterized algorithms for many NP-hard problems [2], [3], [11], [12]. In this paper, we are mainly focused on the randomized algorithms for parameterized problems, which the Parameterized  $P_2$ -Packing problem and the Parameterized Co-Path Packing problem are used to illustrate the random methods given in this paper. We firstly give definitions of the above two problems.

Parameterized  $P_2$ -Packing: Given a graph  $G = (V, E)$  and an integer  $k$ , find a set  $S$  of  $P_2$ s (a  $P_2$  is a simple path of length two) with size  $k$  in  $G$  such that no two  $P_2$ s from  $S$  have common vertices, or report that no such set exists.

Parameterized Co-Path Packing (PCPP): Given a graph  $G = (V, E)$  and an integer  $k$ , find a subset  $F \subseteq V$  of size at most  $k$  such that each connected component in graph  $G[V \setminus F]$  is a path, or report that no such subset exists.

---

\* This work is supported by the National Natural Science Foundation of China under Grant (61232001, 61103033, 61173051), Postdoc Foundation of Central South University.

For the  $P_2$ -Packing problem, Hassin and Rubinstein [10] gave an approximation algorithm with ratio  $35/67$  for the maximum  $P_2$ -Packing problem. De Bontridder *et al.* [5] gave an approximation algorithm with ratio  $2/3$ . For a general subgraph  $H$ , Fellows *et al.* [6] presented an algorithm of running time  $O^*(2^{O(|H|k \log k + k|H| \log |H|)})$  for the  $H$ -Packing problem.<sup>1</sup> Prieto and Sloper [13] gave a parameterized algorithm of time  $O^*(39.43^k)$  for the Parameterized  $P_2$ -Packing problem. Henning *et al.* [7] gave an improved parameterized algorithm with running time  $O^*(14.67^k)$ . Feng *et al.* [8] presented a parameterized algorithm with running time  $O^*(8^k)$  for the Parameterized  $P_2$ -Packing problem, which is the current best result.

The Parameterized Co-Path Packing problem has important application in Bioinformatics [4]. From approximation algorithm point of view, an approximation algorithm with ratio 2 in [9] can be applied to solve the Minimum Co-Path Packing problem. Recently, Chen *et al.* [1] gave a kernel of size  $32k$  for the Parameterized Co-Path Packing problem, and presented an algorithm of running time  $O^*(3.24^k)$ .

In this paper, for the Parameterized  $P_2$ -Packing problem, by using a different way to partition the vertices in the instance, a simple randomized algorithm of running time  $O^*(6.75^k)$  is given, which improves the current best result  $O^*(8^k)$ . For the Parameterized Co-Path Packing problem, the vertices with different degrees are studied. Especially, a randomized algorithm is given for the Parameterized Co-Path Packing problem in degree-bounded graph. By applying iterative compression technique, a randomized algorithm of running time  $O^*(3^k)$  is given, which improves the current best result  $O^*(3.24^k)$ .

## 2 Randomized Algorithm for Parameterized $P_2$ -Packing

For a  $P_2$   $l = (x, y, z)$ ,  $x, z$  are called the End-vertices of  $l$ , and  $y$  is called the Mid-vertex of  $l$ . For a set  $\mathcal{P}$  of  $P_2$ s, if no two  $P_2$ s in  $\mathcal{P}$  have common vertices, then  $\mathcal{P}$  is called a  $P_2$ -Packing in  $G$ . In order to solve the Parameterized  $P_2$ -Packing problem efficiently, we introduce the following problem.

**Constrained  $P_2$ -Packing on Bipartite Graphs:** Given a bipartite graph  $B = (L \cup R, E)$  and an integer  $k$ , either construct a  $P_2$ -Packing  $\mathcal{P}$  of size  $k$  with End-vertices in  $L$ , or report that no such packing exists.

Since the weighted version of Constrained  $P_2$ -Packing on Bipartite Graphs can be solved in polynomial time [8], by assigning each edge in  $B$  with weight 1, the Constrained  $P_2$ -Packing on Bipartite Graphs problem can also be solved in polynomial time.

For a given instance  $(G, k)$  of the Parameterized  $P_2$ -Packing problem, assume that  $\mathcal{P}$  is a  $P_2$ -Packing of size  $k$  in  $G$ . Then, the  $P_2$ s in  $\mathcal{P}$  have  $k$  Mid-vertices and  $2k$  End-vertices, denoted by  $H_1, H_2$  respectively. We want to partition the

---

<sup>1</sup> Following a recent convention, for a function  $f$ , we will use the notion  $O^*(f)$  for the bound  $O(f \cdot n^{O(1)})$ .

vertices in  $V$  into two parts  $V_1$ ,  $V_2$ , such that  $H_1$  is contained in  $V_1$ , and  $H_2$  is contained in  $V_2$ . We give the following random strategy to divide the vertices in  $V$ . For any vertex  $v$  in  $V$ , put  $v$  into  $V_1$  with probability  $2/3$ , and put  $v$  into  $V_2$  with probability  $1/3$ . After deleting all the edges with two ends in  $V_1$  or  $V_2$ , an instance  $B = (V_1 \cup V_2, E')$  of the Constrained  $P_2$ -Packing on Bipartite Graphs problem can be obtained, where  $E'$  is the set of edges with one end in  $V_1$  and the other end in  $V_2$ , which can be solved in polynomial time. The specific random algorithm solving the Parameterized  $P_2$ -Packing problem is given Figure 1.

**Algorithm RP2P( $G, k$ )**

Input: a graph  $G$ , and an integer  $k$

Output: a  $P_2$ -Packing of size  $k$ , or report no such packing exists.

1. **loop**  $c \cdot 6.75^k$  times
- 1.1  $V_1 = V_2 = \emptyset$ ;
- 1.2 **for** each vertex  $v$  in  $V$  **do**
  - put  $v$  into  $V_1$  with probability  $2/3$ , and put  $v$  into  $V_2$  with probability  $1/3$ ;
- 1.3 delete all the edges with two ends either both in  $V_1$  or both in  $V_2$ ;
- 1.4 a bipartite graph  $B = (V_1 \cup V_2, E')$  can be constructed, where  $E'$  is the set of edges with one end in  $V_1$  and the other end in  $V_2$ ;
- 1.5 construct a  $P_2$ -Packing  $\mathcal{P}$  of size  $k$  in  $B$  with End-points in  $V_1$ ;
- 1.6 if step 1.5 is successful, then return  $\mathcal{P}$ ;
2. return("no such packing exists").

**Fig. 1.** A randomized algorithm for the Parameterized  $P_2$ -Packing problem

**Theorem 1.** *The Parameterized  $P_2$ -Packing problem can be solved in time  $O^*(6.75)^k$  with probability larger than  $1 - (1/e)^c$ .*

*Proof.* If the given instance  $(G, k)$  is a No-instance for the Parameterized  $P_2$ -Packing problem, then no matter how the vertices in  $V$  are partitioned, a  $P_2$ -Packing of size  $k$  cannot be found in  $B$ . Thus, step 2 will correctly return that graph  $G$  contains no  $P_2$ -Packing of size  $k$ .

Now assume that  $\mathcal{P}$  is a  $P_2$ -Packing of size  $k$  in  $G$ . Then there are  $2k$  End-vertices and  $k$  Mid-vertices contained in  $\mathcal{P}$ . Let  $H_1$ ,  $H_2$  be the sets of End-vertices, Mid-vertices of  $\mathcal{P}$  respectively,  $|H_1| = 2k$ , and  $|H_2| = k$ . If the vertices in  $\mathcal{P}$  can be partitioned correctly, i.e., all the vertices in  $H_1$  are put into  $V_1$  and all the vertices in  $H_2$  are put into  $V_2$ , then a bipartite graph  $B = (V_1 \cup V_2, E')$  containing a  $P_2$ -Packing of size  $k$  can be constructed in step 1.4. Then, the  $P_2$ -Packing found in step 1.5 can be correctly returned by step 1.6.

We now analyze the probability that the algorithms fails in finding a  $P_2$ -Packing of size  $k$  in  $G$ . Since each vertex in  $V$  is put into  $V_1$ ,  $V_2$  with probability  $2/3$ ,  $1/3$  respectively, for each iteration of step 1, the vertices in  $H_1$ ,  $H_2$  can be correctly partitioned in step 1.2 with probability  $(2/3)^{2k}(1/3)^k = (4/27)^k$ .

Then, the probability that the vertices in  $H_1$ ,  $H_2$  are not correctly partitioned in each iteration is  $1 - (4/27)^k$ . Therefore, the probability that  $H_1$  and  $H_2$  are not partitioned correctly in  $c \cdot 6.75^k$  iterations is  $(1 - (4/27)^k)^{c \cdot 6.75^k} = ((1 - 1/6.75^{k/6})^{6 \cdot 75^k})^c \leq (1/e)^c$ . Therefore, the vertices in  $H_1$ ,  $H_2$  can be partitioned into  $V_1$ ,  $V_2$  respectively with probability larger than  $1 - (1/e)^c$ .

At last, we analyze the time complexity of algorithm **RP2P**. Step 1.2 can be done in  $O(n)$  time, where  $n$  is the number of vertices in  $G$ . It takes  $O(n + m)$  time to do steps 1.3, 1.4, where  $m$  is the number of edges in  $G$ . For step 1.5, a  $P_2$  Packing of size  $k$  in  $G$  can be found in  $O(k(m' + n \log n))$  [8], where  $m'$  is the number of edges in  $B$ . Therefore, the running time of algorithm **RP2P** is  $O(6.75^k k(m' + n \log n)) = O^*(6.75^k)$ .  $\square$

### 3 Randomized Algorithm for Parameterized Co-path Packing Problem

For a vertex  $v$ , let  $N(v)$  denote the neighbors of vertex  $v$ , i.e.,  $N(v) = \{u | (u, v) \in E\}$ , and let  $N[v] = N(v) \cup \{v\}$ . For a graph  $G = (V, E)$ , and a subset  $V' \subseteq V$ , let  $G[V']$  denote the subgraph induced by the vertices in  $V'$ . In order to solve the problem efficiently, we first study the Parameterized Co-Path Packing problem in degree bounded graph.

**Degree-Bounded Parameterized Co-Path Packing (DBPCP):** Given a graph  $G = (V, E)$  and an integer  $k$ , where each vertex  $v$  in  $V$  has degree at most three, find a subset  $F \subseteq V$  of size at most  $k$  such that each connected component in graph  $G[V \setminus F]$  is a path, or report that no such subset exists.

#### 3.1 Kernelizaiton Algorithm for DBPCP

For the Degree-Bounded Parameterized Co-Path Packing problem, some structure properties can be obtained, as follows.

**Lemma 1.** *Given an instance  $(G, k)$  of Degree-Bounded Parameterized Co-Path Packing problem, the number of vertices with degree three in  $G$  is bounded by  $4k$ .*

Given an instance  $(G, k)$  of Degree-Bounded Parameterized Co-Path Packing problem, for a path  $P = \{v_1, v_2, \dots, v_{h-1}, v_h\}$  in  $G$ , if the vertices in  $\{v_2, \dots, v_{h-1}\}$  have degree two and the degrees of  $v_1, v_h$  are not two, then path  $P$  is called a *degree-two-path*.

**Lemma 2.** *For a degree-two-path  $P = \{v_1, v_2, \dots, v_{h-1}, v_h\}$  in  $G$ , if the two vertices  $v_1, v_h$  both have degree three and  $h > 4$ , then edges in  $\{(v_i, v_{i+1}) | i = 2, 3, \dots, h-2\}$  can be contracted; if one vertex of  $\{v_1, v_h\}$  has degree one and the other has degree three, then edges in  $\{(v_i, v_{i+1}) | i = 1, 2, \dots, h-2\}$  can be contracted.*

Based on Lemma 2, we can get the following reduction rule.

**Rule 1.** For a degree-two-path  $P = \{v_1, v_2, \dots, v_{h-1}, v_h\}$  in  $G$ , if the two vertices  $v_1, v_h$  both have degree three and  $h > 4$ , then contract the edges in  $\{(v_i, v_{i+1}) | i = 2, 3, \dots, h-2\}$ . If one vertex of  $\{v_1, v_h\}$  has degree one and the other has degree three, then contract the edges in  $\{(v_i, v_{i+1}) | i = 1, 2, \dots, h-2\}$ .

Let  $G' = (V', E')$  be the graph obtained after applying reduction Rule 1.

**Lemma 3.** In the graph  $G'$ , the number of vertices with degree two is bounded by  $12k$ .

**Lemma 4.** For any vertex  $v$  with degree three in  $G'$ , if there are two vertices  $u, w$  with degree one in  $N(v)$ , then the vertex in  $N(v) \setminus \{u, w\}$  can be deleted.

Based on Lemma 4, we can get the following reduction rule.

**Rule 2.** For any vertex  $v$  with degree three in  $G'$ , if there are two vertices  $u, w$  with degree one in  $N(v)$ , then delete the vertex in  $N(v) \setminus \{u, w\}$ ,  $V' = V' \setminus \{u, v, w\}$ ,  $k = k - 1$ .

Let  $G'' = (V'', E'')$  be the graph obtained after applying reduction Rule 2.

**Lemma 5.** In the graph  $G''$ , the number of vertices with degree one is bounded by  $4k$ .

By applying reduction Rule 1 and Rule 2 repeatedly, a kernel of size  $20k$  for the Degree-Bounded Parameterized Co-Packing problem can be obtained.

**Theorem 2.** The Degree-Bounded Parameterized Co-Path Packing problem admits a kernel of size  $20k$ .

### 3.2 Randomized Algorithm for DBPCP

Assume that  $(G = (V, E), k)$  is a reduced instance of the Degree-Bounded Parameterized Co-Path Packing problem by repeatedly applying reduction Rule 1 and Rule 2, and assume that  $F (|F| \leq k)$  is a solution of the Degree-Bounded Parameterized Co-Path Packing problem. Let  $E'$  be the set of edges in  $E$  with at least one endpoint having degree three. The edges in  $E'$  can be divided into two parts  $A, B$  such that  $A \cup B = E'$ , and for each edge  $e$  in  $A$ , at least one endpoint of  $e$  is contained in  $F$ , and for each edge  $e'$  in  $B$ , no endpoint of  $e$  is contained in  $F$ .

**Lemma 6.** In the reduced graph  $G$ , for the sets  $A, B$ ,  $|A|/|B| \geq 1/2$ .

**Lemma 7.** For an edge  $e = (u, v) \in B$ , and for any vertex  $x$  from  $\{u, v\}$  with degree three, then at least one vertex in  $N(x) \setminus (\{u, v\} \setminus \{x\})$  is contained in  $F$ .

The general idea to randomly solve the Degree-Bounded Parameterized Co-Path Packing problem is as follows. Arbitrarily choose an edge  $e$  from  $E'$ . Then, with probability  $|A|/|E'|$ , the edge  $e$  is from  $A$ , and with probability  $|B|/|E'|$ , edge  $e$  is from  $B$ . If the edge  $e$  is from  $A$ , then at least one endpoint of  $e$  is contained

**Algorithm R-DBPCP( $G, k$ )**Input: a graph  $G$ , and an integer  $k$ Output: a subset  $F \subseteq V$  of size at most  $k$  such that each component in  $G[V \setminus F]$  is a path, or report no such subset exists.

```

1. for each  $k_1, k_2$  with  $k_1 + k_2 \leq k$  do
2.   loop  $c \cdot 2^{k_1}$  times
2.1    $F = C = D = \emptyset$ ;
2.2   let  $E'$  be the set of edges in  $E$  with at least one endpoint having degree three;
2.3   while  $E'$  is not empty do
2.4     randomly choose an edge  $e = (u, v)$  from  $E'$ ;
2.5     put  $e$  into  $C$  with probability  $1/2$ , and put  $e$  into  $D$  with probability  $1/2$ ;
2.6     if  $e$  is contained in  $C$  then
2.7       randomly choose a vertex from  $\{u, v\}$  to put into  $F$ ;
2.8     if  $e$  is contained in  $D$  then
2.9       find a vertex  $w$  from  $\{u, v\}$  with degree three;
2.10      randomly choose a vertex  $y$  from  $N(w) \setminus (\{u, v\} \setminus \{w\})$  to put into  $F$ ;
2.11      let  $E'$  be the set of edges in  $G[V \setminus F]$  with at least one endpoint having degree three;
2.12    if  $|F| \leq k_1$  then
2.13      denote the remaining graph by  $G'$ ;
2.14      if the number of cycles in  $G'$  is at most  $k_2$  then
2.15        for each cycle  $C$  in  $G'$  do
2.16          delete a vertex  $v'$  from  $C$ , and add  $v'$  to  $F$ ;
2.17        return( $F$ ); break;
3.    return("no such subset exists").

```

**Fig. 2.** A randomized algorithm for DBPCP problem

in  $F$ . Randomly choose an endpoint of  $e$  to put into  $F$ . If the edge  $e$  is from  $B$ , find an endpoint  $x$  of  $e$  with degree three, and randomly choose a vertex from  $N(x) \setminus (\{u, v\} \setminus \{x\})$  to put into  $F$ . The specific randomized algorithm solving the Degree-Bounded Parameterized Co-Path Packing problem is given in Figure 2.

**Theorem 3.** *The Degree-Bounded Parameterized Co-path Packing problem can be solved randomly in time  $O^*(2^k)$ .*

*Proof.* First note that if the input instance is a no-instance, step 2 could not find a subset  $F \subseteq V$  with size at most  $k$  such that in graph  $G[V \setminus F]$ , each component is a path, which is rightly handled by step 3.

Now suppose that a subset  $F \subseteq V$  can be found in  $G$  such that each component is a path in  $G[V \setminus F]$ . Then, there must exist two subsets  $F', F'' \subseteq F$  ( $F' \cup F'' = F$ ) such that in  $G[V \setminus F']$ , each vertex has degree at most two, and after deleting the edges in  $F''$ , all the cycles in  $G[V \setminus F']$  are destroyed. Thus, there must exist  $k_1, k_2$  with  $k_1 + k_2 \leq k$  such that  $|F'| = k_1, |F''| = k_2$ .

By arbitrarily choose an edge  $e$  from  $E'$ , with probability  $|A|/|E'|$ , the edge  $e$  is from  $A$ , and with probability  $|B|/|E'|$ , edge  $e$  is from  $B$ . By Lemma 7, for the sets  $A, B$ ,  $|A|/|B| \geq 1/2$ . Therefore, the probability that edge  $e$  is from  $A$  is at least  $1/2$ . In step 2.5, edge  $e$  is put into  $C$  with probability  $1/2$ , and is put into  $D$  with probability  $1/2$ . Since at least one endpoint of edge  $e$  has degree three, at least one vertex from  $N[u] \cup N[v]$  is contained in  $F$ . Assume that  $\{v_1, \dots, v_i\}$  ( $1 \leq i$ ) is the subset of vertices from  $N[u] \cup N[v]$ , which are contained in  $F$ . We now prove that by steps 2.5-2.10, one vertex from  $\{v_1, \dots, v_i\}$  ( $1 \leq i$ ) can be rightly added into  $F$  with probability  $1/2$ . If the edge  $e$  picked in step 2.4 is from  $A$ , then with probability  $1/2$ ,  $e$  is put into  $C$ . Without loss of generality, assume that  $F \cap \{u, v\}$  is  $\{u\}$ . Then, in step 2.7, for the vertices  $\{u, v\}$ ,  $u$  can be rightly added into  $F$  with probability  $1/2$ . Thus, if the edge  $e$  picked in step 2.4 is from  $A$ , then with probability  $1/4$ ,  $u$  can be rightly added into  $F$  in step 2.7. On the other hand, if the edge  $e$  picked in step 2.4 is from  $B$ , then with probability  $1/2$ ,  $e$  is put into  $D$  in step 2.5. In this case, no vertex from  $\{u, v\}$  is contained in  $F$ . Without loss of generality, assume that vertex  $u$  has degree three. Consequently, at least one vertex from  $N(u) \setminus \{v\}$  is added into  $F$ . Assume that vertex  $x$  from  $N(u) \setminus \{v\}$  is contained in  $F$ . Then, in step 2.10, for the vertices in  $N(u) \setminus \{v\}$ , vertex  $x$  can be rightly added into  $F$  with probability  $1/2$ . Therefore, if the edge  $e$  picked in step 2.4 is from  $B$ , then with probability  $1/4$ , vertex  $x$  is rightly added into  $F$ . Thus, for the edge  $e$  picked in step 2.4, the probability that at least one vertex in  $\{v_1, \dots, v_i\}$  is rightly put into  $F$  is  $1/4 + 1/4 = 1/2$ . Then, the probability that all vertices in  $F'_1$  are deleted is at least  $(1/2)^{k_1}$ . Therefore, the probability that the vertices in  $F'_1$  are not rightly deleted is  $1 - (1/2)^{k_1}$ . Therefore, after  $c \cdot 2^{k_1}$  operations, none of the executions of steps 2.1-2.11 can rightly handle the vertices in  $F'_1$  is  $(1 - (1/2)^{k_1})^{c \cdot 2^{k_1}} = ((1 - (1/2)^{k_1})^{2^{k_1}})^c \leq (1/e)^c$ . Therefore, after  $c \cdot 2^{k_1}$  operations, the algorithm can correctly handle the vertices in  $F'_1$  with probability larger than  $1 - (1/e)^c$ .

For each loop in step 2, if  $|F| \leq k_1$  in step 2.12 and there exist cycles in the remaining graph, the cycles can be destroyed by choosing any vertex in the cycle.

Step 2.2 can be done in time  $O(n+m)$ , and step 2.3 can be done in  $O(m(m+n))$ , where  $m, n$  are the number of edges and vertices in graph  $G$  respectively. Step 2.12-2.16 can be done in time  $O(n+m)$ . Therefore, algorithm R-DBPCP runs in time  $O(2^{k_1}m(n+m)) = O^*(2^k)$ .  $\square$

## 4 Randomized Algorithm for the PCPP Problem

In this section, using iterative compression technique, we give a randomized algorithm of running time  $O^*(3^k)$  for the Parameterized Co-Path Packing problem. We first give the following definitions.

Parameterized Co-Path Packing Compression (PCPPC): Given a graph  $G = (V, E)$ , an integer  $k$ , and a subset  $F \subseteq V$  of size  $k+1$ , where in graph  $G[V \setminus F]$ , each connected component is a path, find a subset  $F' \subseteq V$  of

size at most  $k$  such that each connected component in graph  $G[V \setminus F']$  is a path, or report that no such subset exists.

**Special Parameterized Co-Path Packing Compression (SPCPPC):** Given a graph  $G = (V, E)$ , an integer  $k$ , and a subset  $F \subseteq V$  of size  $k+1$ , where in graph  $G[V \setminus F]$ , each connected component is a path, find a subset  $F' \subseteq V$  of size at most  $k$  such that  $F' \cap F = \emptyset$ , and each connected component in graph  $G[V \setminus F']$  is a path, or report that no such subset exists.

We now give an algorithm solving the Special Parameterized Co-Path Packing Compression problem.

**Lemma 8.** *Given an instance  $(G = (V, E), k, F)$  of the Special Parameterized Co-Path Packing Compression problem, if there exists a vertex  $v$  in  $V \setminus F$  with degree at least five, then  $v$  can be deleted; if there exists a vertex  $v$  in  $V \setminus F$  with degree four and  $|N(v) \cap F| = 3$ , then  $v$  can be deleted.*

For the instance  $(G = (V, E), k, F)$  of the Special Parameterized Co-Path Packing Compression problem, if there exists a vertex  $v$  in  $F$  with degree at least three, then  $v$  is called a *special vertex* in  $F$ .

**Lemma 9.** *For a special vertex  $v$  in  $F$ , if  $|F \cap N(v)| = 2$ , then the vertices in  $N(v) \setminus F$  can be deleted; if  $|F \cap N(v)| = 1$ , then at least  $|N(v) \setminus F| - 1$  vertices in  $N(v) \setminus F$  can be deleted.*

Given an instance  $(G = (V, E), k, F)$  of the Special Parameterized Co-Path Packing Compression problem, assume that  $F'$  is the solution of the problem. In the following, we first give the algorithm to deal with the vertices with degree four in  $V \setminus F$  and having two neighbors in  $V \setminus F$ : for a vertex  $v$  with degree four in  $V \setminus F$  and having two neighbors in  $V \setminus F$ , either  $v$  is contained in  $F$ , or the two vertices in  $N(v) \setminus F$  are contained in  $F$ . The specific algorithm is given in Figure 3.

**Theorem 4.** *Algorithm Bran-V( $G, k_1, F, \emptyset$ ) runs in time  $O(1.62^{k_1} n)$  and returns a collection of at most  $1.62^{k_1}$  sets of vertices, where  $n$  is the number of vertices in  $G$ .*

*Proof.* For an instance  $(G = (V, E), k, F)$  of the Special Parameterized Co-Path Packing Compression problem, assume that  $F'$  is the solution of the problem. For a vertex  $v$  of  $V \setminus F$  with degree four and having two neighbors in  $V \setminus F$ , assume that two neighbors  $u, w$  of  $v$  are contained in  $V \setminus F$ . Either vertex  $v$  is in  $F'$  or the vertices  $u, w$  are in  $F'$ , which corresponds the two branchings in step 4 and step 5 respectively. Assume that  $V'$  is the set of all vertices with degree four in  $G$ , and let  $V'' \subseteq V'$  be the set of vertices in  $\bigcup_{v \in V'} N[v]$  that are contained in  $F'$ . Let  $k_1 = |V''|$  and let  $T(k_1)$  be the size of search tree obtained by calling algorithm Bran-V( $G, k_1, F, Q$ ). It is easy to get the following recurrence relation:  $T(k_1) = T(k_1 - 1) + T(k_1 - 2)$ . Then,  $T(k_1) = 1.62^{k_1}$ . Therefore, algorithm Bran-V( $G, k_1, F, \emptyset$ ) runs in time  $O(1.62^{k_1} n)$  and returns a collection of at most  $1.62^{k_1}$  sets of vertices.  $\square$

**Algorithm Bran-V( $G, k_1, F, Q$ )**

**Input:** a graph  $G = (v, E)$ , an integer  $k$ , a subset  $F$  of vertices, and a subset  $Q$  of vertices

**Output:** the collection of vertices in  $V \setminus F$ , each contains  $k_1$  vertices

1. **if** ( $|Q| > k_1$ ) **then** abort; **else** return  $Q$ ;
2. arbitrarily pick a vertex  $v$  in  $V \setminus F$  with degree four and having two neighbors in  $V \setminus F$ ;
3. let  $u, w$  be the two vertices in  $N(v) \setminus F$ ;
4. Bran-V( $G \setminus \{v\}, k_1, F, Q \cup \{v\}$ );
5. Bran-V( $G \setminus \{u, v\}, k_1, F, Q \cup \{u, v\}$ );

**Fig. 3.** Branching of vertices with degree four

For an instance  $(G = (V, E), k, F)$  of the Special Parameterized Co-Path Packing Compression problem, if all the vertices in  $G$  have degree at most three, the algorithm R-DBPCP can be modified slightly to find the solution of the problem.

**Theorem 5.** *For an instance  $(G = (V, E), k, F)$  of the Special Parameterized Co-Path Packing Compression problem, if all the vertices in  $G$  have degree at most three, then a subset  $F'$  of size  $k$  can be found in  $O^*(2^k)$  time with probability larger than  $1 - (1/e)^c$  such that  $F \cap F' = \emptyset$  and each connected component in  $G[V \setminus F']$  is a path.*

The proof of Theorem 5 is similar to the proof of Theorem 3, which is neglected here.

The general idea solving the Special Parameterized Co-Path Packing Compression problem is that: the vertices with degree large than four, and the special vertices are handled firstly. Then, by calling algorithm Bran-V( $G, k_1, F, \emptyset$ ), the vertices with degree four and having two neighbors in  $V \setminus F$  can be dealt with. Then, in the remaining graph, each vertex has degree at most three, which can be handled by algorithm R-DBPCP. The specific algorithm solving the Special Parameterized Co-Path Packing Compression problem is given in Figure 4.

**Theorem 6.** *The Special Parameterized Co-Path Packing Compression problem can be solved randomly in  $O^*(2^k)$  time.*

*Proof.* If the input instance is a no-instance, step 5 could not find a subset  $F' \subseteq V$  with size at most  $k$  such that  $F' \cap F = \emptyset$ , and in graph  $G[V \setminus F']$ , each component is a path, which is rightly handled by step 6.

Now suppose that a subset  $F' \subseteq V$  can be found in  $G$  such that  $F' \cap F = \emptyset$ , and in graph  $G[V \setminus F']$ , each component is a path. Then, there must exist three subsets  $F'_1, F'_2, F'_3$  ( $|F'_1| = k_1, |F'_2| = k_2, |F'_3| = k_3$ ) such that  $F'_1 + F'_2 + F'_3 = F'$  and  $F'_1, F'_2, F'_3$  have the following properties: (1) after deleting the vertices in  $F'_1$ , there does not exist a special vertex in the remaining graph; (2) after deleting the vertices in  $F'_1 \cup F'_2$ , all the vertices in the remaining graph have degree bounded by three.

**Algorithm R-SPCPPC( $G, k, F$ )**

Input: a graph  $G$ , and an integer  $k$

Output: a subset  $F' \subseteq V$  of size at most  $k$  such that  $F \cap F' = \emptyset$  and each component in  $G[V \setminus F']$  is a path, or report no such subset exists.

1.  $F' = \emptyset$ ;
2. **if** there exists a vertex  $v$  in  $V \setminus F$  with degree at least five **then**  
 $F' = F' \cup \{v\}$ ,  $V = V \setminus \{v\}$ ,  $k = k - 1$ ;
3. **if** there exists a vertex  $v$  in  $V \setminus F$  with degree four and  $|N(v) \cap F| = 3$  **then**  
 $F' = F' \cup \{v\}$ ,  $V = V \setminus \{v\}$ ,  $k = k - 1$ ;
4. **if** there exists a special vertex  $v$  with  $|F \cap N(v)| = 2$  **then**  
 $F' = F' \cup (N(v) \setminus F)$ ,  $V = V \setminus (N(v) \setminus F)$ ,  $k = k - |N(v) \setminus F|$ ;
5. **for** each  $k_1, k_2, k_3$  with  $k_1 + k_2 + k_3 \leq k$  **do**
  - 5.1. **loop**  $c \cdot 2^{k_1}$  times
  - 5.2.  $H = H'' = \emptyset$ ;
  - 5.3. let  $V'$  be the set of special vertices such that for each  $v$  in  $V'$ ,  
 $|F \cap N(v)| = 1$ ;
  - 5.4. **while**  $V'$  is not empty **do**
    - 5.5. randomly choose a vertex  $u$  from  $N(v) \setminus F$ , and put  $u$  into  $H$ ;
    - 5.6. let  $V'$  be the set of special vertices in  $G[V \setminus (F \cup H)]$  such that for  
each  $v$  in  $V'$ ,  $|F \cap N(v)| = 1$ ;
    - 5.7. **if**  $|H| \leq k_1$  **then**
      - 5.8. call algorithm Bran-V( $G[V \setminus H], k_2, F, \emptyset$ );
      - 5.9. **for** each set  $H'$  returned by algorithm Bran-V( $G, k_2, F, \emptyset$ ) **do**
        - 5.10. let  $G' = (V', E') = G[V \setminus (H \cup H')]$ ;
        - 5.11. let  $H''$  be the set obtained by calling algorithm R-DBPCP;
        - 5.12. **if**  $|H| + |H'| + |H''| \leq k$  **then**
          - 5.13.  $F' = F' \cup (H \cup H' \cup H'')$ ;
      - 5.13. return( $F'$ ); stop the loop in step 5.1;
    6. return("no such subset exists").

**Fig. 4.** A randomized algorithm for SPCPPC problem

The correctness of steps 2-4 can be obtained directly by Lemma 8 and Lemma 9. For a special vertex  $v$  with  $|F \cap N(v)| = 1$ , by Lemma 9, at least  $|N(v) \setminus F| - 1$  vertices in  $N(v) \setminus F$  can be deleted. Assume that  $T \subseteq N(v) \setminus F$  are the subset of vertices contained in  $F'$ . Since the degree of  $v$  is at least three, by randomly choosing a vertex  $u$  from  $N(v) \setminus F$ , the probability that  $u$  is from  $T$  is at least  $1/2$ . Then, the probability that all vertices in  $F'_1$  are deleted is at least  $(1/2)^{k_1}$ , i.e., when steps 5.4-5.6 are done, all the special vertices are destroyed with probability  $(1/2)^{k_1}$ . In step 5.8, algorithm Bran-V( $G[V \setminus H]$ ,  $k_2$ ,  $F$ ,  $\emptyset$ ) is called to deal with the vertices with degree four and  $|N(v) \cap (V \setminus F)| = 2$ . By Theorem 4, a collection of at most  $1.62^{k_1}$  sets of vertices can be returned. Then, in step 5.10, there does not exist a vertex with degree four. Therefore, algorithm R-DBPCP can be used to find a subset  $F'_3 \subseteq V'$  such that  $F'_3 \cap F = \emptyset$  and each connected component in  $G'[V' \setminus F'_3]$  is a path. By Theorem 5, the subset  $F'_3$  can be found with probability larger than  $1 - (1/e)^c$ . Therefore, in step 5.12, if  $|H| + |H'| + |H''| \leq k$ , then a solution  $F'$  can be returned such that  $F' \cap F = \emptyset$  and each connected component in  $G[V \setminus F']$  is a path. For the step 5.1, the probability that the vertices in  $F'_1$  are not rightly deleted is  $1 - (1/2)^{k_1}$ . Therefore, after  $c \cdot 2^{k_1}$  operations, the probability that none of the executions of steps 5.5-5.6 can rightly handle the vertices in  $F'_1$  is  $(1 - (1/2)^{k_1})^{c \cdot 2^{k_1}} = ((1 - (1/2)^{k_1})^{2^{k_1}})^c \leq (1/e)^c$ . Therefore, after  $c \cdot 2^{k_1}$  operations, the algorithm can correctly handle the vertices in  $F'_1$  with probability larger than  $1 - (1/e)^c$ .

Now we analyze the running time of algorithm R-SPCPCC( $G, k, F$ ). Steps 2-4 can be done in  $O(n+m)$  time. The while loop in step 5.4 can be done in  $O(n(n+m))$  time. By Theorem 4, algorithm Bran-V( $G, k_1, F, \emptyset$ ) runs in time  $O(1.62^{k_1}n)$ , and by Theorem 5, algorithm R-DBPCP can be done in time  $O^*(2^{k_3})$ . Therefore, step 5 can be done in  $O(2^{k_1}1.62^{k_2}2^{k_3}k^3(n+m)) = O(2^{k_1+k_2+k_3}k^3(n+m)) = O^*(2^k)$ .  $\square$

Based on the algorithm solving the Special Parameterized Co-Path Packing Compression problem, the Parameterized Co-Path Packing Compression problem can be solved.

**Theorem 7.** *The Parameterized Co-Path Packing Compression problem can be solved randomly in  $O^*(3^k)$  time.*

Based on the iterative compression technique, the Parameterized Co-Path Packing problem can be solved.

**Theorem 8.** *The Parameterized Co-Path Packing problem can be solved randomly in  $O^*(3^k)$  time.*

## 5 Conclusion

In this paper, we study the randomized techniques for the parameterized problems. For the  $P_2$ -Packing problem, a randomized algorithm of running time  $O^*(6.75^k)$  is given, improving the current best result  $O^*(8^k)$ . For the Parameterized Co-Path Packing problem, a randomized algorithm of running time

$O^*(3^k)$  is given, improving the current best result  $O^*(3.24^k)$ . How to apply the randomized methods in this paper to solve other parameterized problems is an interesting topic, which is also our future research.

## References

1. 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)
2. Chen, J., Lu, S.: Improved parameterized set splitting algorithms: A probabilistic approach. *Algorithmica* 54(4), 472–489 (2008)
3. Chen, J., Lu, S., Sze, S.H., Zhang, F.: Improved algorithms for path, matching, and packing problems. In: Proc. of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), pp. 298–307 (2007)
4. Chauve, C., Tannier, E.: A methodological framework for the reconstruction of contiguous regions of ancestral genomes and its application to mammalian genome. *PLoS Comput. Biol.* 4, e1000234 (2008)
5. De Bontridder, K., Halldórsson, B., Lenstra, J., Ravi, R., Stougie, L.: Approximation algorithms for the test cover problem. *Math. Program., Ser. B* 98, 477–491 (2003)
6. Fellows, M., Heggernes, P., Rosamond, F., Sloper, C., Telle, J.A.: Finding  $k$  disjoint triangles in an arbitrary graph. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 235–244. Springer, Heidelberg (2004)
7. Fernau, H., Raible, D.: A parameterized perspective on packing paths of length two. *Journal of Combinatorial Optimization* 18(4), 319–341 (2009)
8. Feng, Q., Wang, J., Chen, J.: Matching and  $P_2$ -Packing: Weighted Versions. In: Fu, B., Du, D.-Z. (eds.) COCOON 2011. LNCS, vol. 6842, pp. 343–353. Springer, Heidelberg (2011)
9. Fujito, T.: Approximating node-deletion problems for matroidal properties. *J. Algorithms* 31, 211–227 (1999)
10. Hassin, R., Rubinstein, S.: An approximation algorithm for maximum triangle packing. *Discrete Appl. Math.* 154, 971–979 (2006)
11. Marx, D., Razgon, I.: Fixed-parameter tractability of multicut parameterized by the size of the cutset. In: Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC 2011), pp. 469–478 (2011)
12. Marx, D.: Randomized Techniques for Parameterized Algorithms. In: Thilikos, D.M., Woeginger, G.J. (eds.) IPEC 2012. LNCS, vol. 7535, p. 2. Springer, Heidelberg (2012)
13. Prieto, E., Sloper, C.: Looking at the stars. *Theoretical Computer Science* 351, 437–445 (2006)

# DVS Scheduling in a Line or a Star Network of Processors<sup>\*</sup>

Zongxu Mu and Minming Li

Department of Computer Science, City University of Hong Kong  
`mike.mu@student.cityu.edu.hk, minmli@cs.cityu.edu.hk`

**Abstract.** Dynamic Voltage Scaling (DVS) is a technique which allows the processors to change speed when executing jobs. Most of the previous works either study single processor or multiple parallel processors. In this paper, we consider a network of DVS enabled processors. Every job needs to go along a certain path in the network and has a certain workload finished on any processor it goes through before it moves on to the next processor. Our objective is to minimize the total energy consumption while finishing every job before its deadline. Due to the intrinsic complexity of this problem, we only focus on line networks with two nodes and a simple one-level tree network (a star). We show that in some of these simple cases, the optimal schedule can be computed efficiently and interleaving is not needed to achieve optimality. However, in both types of networks, how to find the optimal sequence of execution remains a big challenge for jobs with general workloads.

## 1 Introduction

Energy efficiency is always a primary concern for chip designers not only for the sake of prolonging the lifetime of batteries which are the major power supply of portable electronic devices but also for the environmental protection purpose when large facilities like data centers are involved. Currently, processors capable of operating at a range of frequencies are already available, such as Intel's Speed-Step technology and AMD's PowerNow technology. The capability of the processor to change voltages is often referred to in the literature as DVS (Dynamic Voltage Scaling) techniques. For DVS processors, since energy consumption is at least a quadratic function of the supply voltage (which is proportional to CPU speed), it saves energy to let the processor run at the lowest possible speed while still satisfying all the timing constraints, rather than running at full speed and then switching to idle.

One of the earliest theoretical models for DVS was introduced by Yao, Demers and Shenker [20] in 1995. They assumed that the processor can run at any speed and each job has an arrival time and a deadline. They gave a characterization of the minimum-energy schedule (MES) and an  $O(n^3)$  algorithm for computing

---

\* This work was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 124411].

it, which was improved to  $O(n^2 \log n)$  in [16]. No special assumption was made on the power consumption function except convexity. Several online heuristics were also considered including the Average Rate Heuristic (AVR) and Optimal Available Heuristic (OPA). Under the common assumption of power function  $P(s) = s^\alpha$ , they showed that AVR has a competitive ratio of  $2^{\alpha-1} \alpha^\alpha$  for all job sets. Thus its energy consumption is at most a constant times the minimum required. Later on, under various related models and assumptions, more algorithms for energy-efficient scheduling have been proposed.

Bansal et al. [4] further investigated the online heuristics for the model proposed by [20] and proved that the heuristic OPA has a tight competitive ratio of  $\alpha^\alpha$  for all job sets. For the temperature model where the temperature of the processor is not allowed to exceed a certain thermal threshold, they showed how to solve it within any error bound in polynomial time. Chan et al. [5] investigated a slightly more realistic model where the maximum speed is bounded. They proposed an online algorithm which is  $O(1)$ -competitive in both energy consumption and throughput. More works on the speed bounded model can be found in [3][6][14]. Ishihara and Yasuura [11] initiated the research on discrete DVS problem where a CPU can only run at a set of given speeds. They solved the case when the processor is only allowed to run at two different speeds. Kwon and Kim [13] extended it to the general discrete DVS model where the processor is allowed to run at speeds chosen from a finite speed set. They gave an algorithm for this problem based on the MES algorithm in [20]. Later on, Li et al. [15] improved the time complexity to  $O(dn \log n)$  where  $d$  is the number of available speeds. When the CPU can only change speed gradually instead of instantly, [9] discussed about some special cases that can be solved optimally in polynomial time. Recently, Wu et al. [19] showed that the optimal schedule for jobs with agreeable deadlines can be computed in polynomial time. Pruhs et al. [18] introduced profit into DVS scheduling. They assume that the profit obtained from a job is a function on its finishing time and on the other hand money needs to be paid to buy energy to execute jobs. They give a lower bound on how good an online algorithm can be and also give a constant competitive ratio online algorithm in the resource augmentation setting. A survey on algorithmic problems in power management for DVS can be found in [1][10].

Most recently, the idea of combining DVS processors and networks was explored by Andrews et al. [2]. They studied the problem of energy efficient routing in the network. They assumed that each link  $e$  in the network has an associated energy cost function  $f_e(s)$  where  $s$  is the demand routed through  $e$ . The goal is to minimize the total energy consumption on links while delivering all the demands from their respective sources to the destinations. They designed an approximation algorithm with constant ratio when  $f_e(s) = \mu s^\alpha$  where  $\mu$  and  $\alpha$  are fixed constants.

Motivated by the above model, in this paper, we try to explore another combination of DVS processors and networks where each node is a DVS processor and the network is able to handle multiple jobs following different paths in the network.

The remaining paper is organized as follows. In Section 2, we introduce the problem formulation. Section 3 studies line networks and Section 4 studies star networks. Finally, we conclude our paper in Section 5. Due to space limit, some proofs are omitted.

## 2 Problem Formulation

We use a directed graph  $G = (V, E)$  to represent the network of processors. There is a special node  $s$  having directed edges to some or all the other nodes and also a special node  $t$  having directed edges from some or all the other nodes. Every node  $v \in V$  other than  $s$  and  $t$  represents one processor. Edge  $e = (u, v)$  means that jobs can be passed from processor  $u$  to processor  $v$ . There is a set of jobs  $J = \{J_1, J_2, \dots, J_n\}$  to be processed in this network of processors. Every job  $J_i$  is characterized by an arrival time  $r_i$ , a deadline (or delay bound)  $d_i$ , a path  $P_i$  on  $G$ ,  $s \rightarrow v_1^i \rightarrow v_2^i \rightarrow \dots \rightarrow v_{k_i}^i \rightarrow t$  and a set of workloads  $\{w_{i,v_1^i}, \dots, w_{i,v_{k_i}^i}\}$  which means that  $J_i$  needs to go through every processor on the path in sequence and needs  $w_{i,v_m^i}$  CPU cycles on processor  $v_m^i$ ; when  $J_i$  reaches  $t$ , the time should be less than  $d_i$ . To make the formulation more concise, we normalize  $d_i$  so that  $\max d_i = 1$ .

Each processor has the ability to adjust its speed when executing jobs. If the processor continuously executes a job with the requirement of  $C$  CPU cycles at speed  $s$ , then the processor needs  $C/s$  time to finish its responsibility for the job. The most important element in DVS scheduling is the speed-to-power function  $P(s)$  which represents the amount of energy consumed per unit time if the processor runs at speed  $s$ . In real systems,  $P(s)$  is convex and usually has the form  $P(s) = s^\alpha$  where  $\alpha$  is 2 or 3. We want to produce a schedule for all the processors so that every job is following the correct sequence of processors and is finished by its deadline. Among all the feasible schedules, we want to find a schedule with minimum energy consumption. To formalize the problem description, we give the following definition:

A schedule  $S$  for job set  $J$  consists of a schedule for every processor  $v$  consisting of two functions  $(s_v(t), job_v(t))$  which define, respectively, the processor speed and the job being executed at time  $t$  on processor  $v$ .

A schedule is feasible if for every job  $J_k$ , the following conditions are satisfied:

- (1)  $s_v(t) \geq 0$ ;
- (2)  $job_v(t) \in \{1, 2, \dots, n\}$  where  $n$  is the number of jobs in the job set  $J$ ;
- (3)  $\int_{r_i}^{d_i} s_v(t) \delta(k, job_v(t)) dt = w_{k,v}$ , where  $\delta(i, j) = 1$  if  $i = j$  and  $\delta(i, j) = 0$  otherwise;
- (4)  $\max\{t | job_u(t) = k\} \leq \min\{t | job_v(t) = k\}$  if  $u$  is before  $v$  in the path  $P_k$ .

The energy consumption of a schedule  $S = \{s_v(t), job_v(t)\}$  is defined as  $E(S) = \sum_{v \in V} \int_0^1 P(s_v(t)) dt$  with the power function given as  $P(s) = s^\alpha$ . Hence, the optimization problem we study is:

$$\begin{aligned}
& \min \sum_{v \in V} \int_0^1 P(s_v(t)) dt \\
& \text{s.t. } s_v(t) \geq 0; \\
& \quad job_v(t) \in \{1, 2, \dots, n\} \\
& \quad \int_{r_i}^{d_i} s_v(t) \delta(k, job_v(t)) dt = w_{kv}; \\
& \quad \max\{t | job_u(t) = k\} \leq \min\{t | job_v(t) = k\} \text{ if } u \text{ is before } v \text{ in the path } P_k
\end{aligned}$$

Since now the arrival time of a job at a certain processor is affected by how fast it is done on previous processors, the energy minimization problem is not simply the combination of single processor scheduling problems. New insights need to be identified in order to solve this problem. In the following discussion, we assume  $\alpha = 2$ .

### 3 Line Networks

We start by investigating a special case when the network of processors forms a line (called line network). In other words, the graph is  $G = (V, E)$  where  $V = \{s, v_1, v_2, \dots, v_m, t\}$  and  $E = \{(s, v_1), (v_m, t), (v_i, v_{i+1}) | 1 \leq i \leq m-1\}$  and  $m > 1$ . Every job needs to go through the only path in graph  $G$ .

In the problem, we require all jobs to be processed and pass through the network within the period  $[0, d]$ . More precisely, we have  $r_k = 0$  and  $d_k = d$  for all  $J_k \in J$ . To simplify the notation, we use  $j$  to represent  $v_j^i$ , which is the  $j$ th node in the line network. Thus,  $w_{i,j}$  represents the workload of job  $i$  on processor  $j$ . Under the above setting, it is easy to see that the optimal schedule will use only one speed when executing  $w_{i,j}$ . Therefore, we use  $s_{i,j}$  to represent the processing speed of  $w_{i,j}$ ,  $t_{i,j}$  to represent the time spent on processing  $w_{i,j}$  and  $E_{i,j}$  to represent the energy consumption of executing  $w_{i,j}$ . Thus, the total energy consumption is  $\sum_{i,j} E_{i,j}$  (we will use  $E$  to represent it afterward), which is our target to minimize.

The line network scheduling problem in our model resembles flowshop scheduling model with an extra twist of speed change. A flow shop is an environment where multiple operations required for a job have to be done in the same order [17]. In other words, the jobs have to follow the same route of machines. In the makespan flowshop scheduling model, each job needs to pass  $L$  stages in order, with each stage consisting of  $M^{(l)}$  processors, and the objective is to minimize the makespan of all jobs. As presented by Johnson in [12] which is widely known as Johnson's Rule, the flowshop scheduling problem can be solved to optimality when  $L = 2$ . However, with  $L > 2$ , such optimality is no longer achievable. Indeed, Garey et al. have proved that optimal scheduling is already NP-hard with  $L = 3$  [7], i.e., the  $F3||C_{max}$  problem, using the standard notation proposed by Graham et al. [8].

However, our model is indeed different from makespan flowshop scheduling because the processing time of a job on one processor is not fixed due to processor's ability to change speed and our objective is minimizing the energy consumption instead of minimizing makespan. For the makespan flowshop model, a processor

may be idle in the middle of an optimal schedule; however, it is not the case for the model we study because the energy consumption can always be reduced by extending the processing time of one or both of the jobs before and after the idle period. Therefore, it may be necessary to operate at different speeds in order to achieve energy minimization. This imposes further complexity to the problem.

In our model, a schedule is called an optimal schedule (or min-energy schedule) if its energy consumption is the minimum among all the feasible schedules. We are interested in efficient scheduling algorithms where given a job set  $J$  and a network of processors represented by  $G$ , we aim to find a schedule  $S$  with the minimum energy consumption while achieving the delay bound of every job.

We first define non-interleaving schedules and interleaving schedules.

**Definition 1.** A non-interleaving schedule is a schedule satisfying:

1. For all  $i' (1 \leq i' \leq n)$ , if  $w_{i',1}$  is the  $j$ th finished workload among all  $w_{i,1}$ , then  $w_{i',2}$  is also the  $j$ th finished workload among all  $w_{i,2}$ . It guarantees that all jobs will be processed in the same order on both processors.
2. Once  $w_{i,j}$  starts being processed, the  $j$ th processor will not process any other workload until  $w_{i,j}$  is completed.

A schedule violating any of the two conditions is called an interleaving schedule.

The second property implies non-preemption in flowshop scheduling. Together with the first property it leads to permutation in the flowshop.

### 3.1 Line Network of 2 Processors with 2 Jobs

In this subsection, we will find the optimal schedule for 2 jobs in a line network with 2 processors. Firstly, we argue that we can always find an optimal schedule among non-interleaving schedules. The proof is given in Section 3.2 where we discuss the more general case of energy minimization in a line network of 2 processors with  $n$  jobs. Here, we still deal with 2-job case separately for the purpose of elaborating the proving techniques we use for other cases.

If the schedule begins with job  $J_1$ , the workload  $w_{1,1}$  will be the first to be processed. With no interleaving, the workloads  $w_{i,j}$  ( $i = 1, 2$ ;  $j = 1, 2$ ) will be processed as illustrated by Table 1.

**Table 1.** Work flow in the case of 2 jobs and 2 processors

Node 1	Node 2
$w_{1,1}$	
$w_{2,1}$	$w_{1,2}$
	$w_{2,2}$

Note that  $w_{1,1}$  must finish before  $w_{1,2}$  and  $w_{2,1}$ , while  $w_{2,2}$  can only be processed after  $w_{2,1}$  and  $w_{1,2}$  are finished. In addition,  $w_{2,1}$  and  $w_{1,2}$  will start and

stop processing at the same time. Otherwise, it is always possible to reduce the energy consumption by extending the one with the later start time or earlier end time, because this will not affect the execution of  $w_{1,1}$  and  $w_{2,2}$ , but will increase the processing time, hence reduce the processing speed of  $w_{2,1}$  or  $w_{1,2}$ .

Note that the speed setting problem is in essence equivalent to a time division problem, where each job is executed with the minimum possible constant speed within its assigned interval. With a workload  $w$  executed in a time interval with length  $t$  at a fixed speed, the total energy consumption is  $s^2t = \frac{w^2}{t^2}t = \frac{w^2}{t}$ .

Therefore, we assume  $w_{1,1}$  takes a period of  $t_1$ ,  $w_{1,2}$  and  $w_{2,1}$  takes a period of  $t_2$ , and  $w_{2,2}$  takes a period of  $t_3$ , where  $t_1 + t_2 + t_3 = d$ . Then, the total energy consumption is

$$E = \frac{w_{1,1}^2}{t_1} + \frac{w_{1,2}^2 + w_{2,1}^2}{t_2} + \frac{w_{2,2}^2}{t_3}.$$

We remark that  $w_{1,1}$  and  $w_{2,2}$  should be done at the same constant speed within the period  $d - t_2$  in the optimal schedule. Otherwise, we can reduce the power consumption by speeding up the originally slower one and slowing down the other. Referring to the deduction above, the total energy consumption is

$$E = \frac{(w_{1,1} + w_{2,2})^2}{d - t_2} + \frac{w_{1,2}^2 + w_{2,1}^2}{t_2}.$$

Therefore, to minimize  $E$  subject to  $0 < t_2 < d$ , we need to have

$$\frac{dE}{dt_2} = \frac{(w_{1,1} + w_{2,2})^2}{(d - t_2)^2} - \frac{w_{1,2}^2 + w_{2,1}^2}{t_2^2} = 0.$$

Therefore, we get

$$t_2 = \frac{d\sqrt{w_{1,2}^2 + w_{2,1}^2}}{w_{1,1} + \sqrt{w_{1,2}^2 + w_{2,1}^2} + w_{2,2}}.$$

Because

$$\frac{d^2E}{dt_2^2} = \frac{2(w_{1,1} + w_{2,2})^2}{(d - t_2)^3} + \frac{2(w_{1,2}^2 + w_{2,1}^2)}{t_2^3} > 0,$$

for the schedule proposed in Table 1, we can obtain the minimized value of  $E$ :

$$E_{\min} = \frac{\left(w_{1,1} + \sqrt{w_{1,2}^2 + w_{2,1}^2} + w_{2,2}\right)^2}{d}.$$

Interestingly, the calculated  $E_{\min}$  above is exactly the minimum energy consumption in the situation where

- There are 3 processors in the line network, and
- There is 1 job with workloads  $w_{1,1}$ ,  $w_0$  and  $w_{2,2}$  respectively on three processors where  $w_0 = \sqrt{w_{1,2}^2 + w_{2,1}^2}$ .

Similarly, if we begin with job 2, the minimum energy consumption will be

$$\frac{\left(w_{2,1} + \sqrt{w_{2,2}^2 + w_{1,1}^2} + w_{1,2}\right)^2}{d}.$$

Hence, whether having  $J_1$  or  $J_2$  done first will lead to the energy minimization is determined by the relative value of  $w_{2,1} + \sqrt{w_{2,2}^2 + w_{1,1}^2} + w_{1,2}$  and  $w_{1,1} + \sqrt{w_{1,2}^2 + w_{2,1}^2} + w_{2,2}$ .

### 3.2 Line Network of 2 Processors with n Jobs

**Lemma 1.** *In a line network of 2 processors with  $n$  jobs, an optimal schedule can always be found among non-interleaving schedules.*

*Proof.* We prove the lemma by showing that it is always possible to turn an interleaving schedule into a non-interleaving schedule without increasing the energy consumption. Therefore, if an interleaving schedule can achieve energy minimization, the minimized energy can be achieved by a corresponding non-interleaving schedule as well.

We number the jobs in a way that  $w_{i,1}$  finishes earlier than  $w_{j,1}$  for all  $i < j$ .

Firstly, we consider the schedule on the second processor. For any pair of  $w'_{j,2}$  and  $w'_{i,2}$  which denote adjacent segments where  $i < j$  and  $w'_{j,2}$  is processed before  $w'_{i,2}$ , it is always possible to change their order into  $w'_{i,2}$  and then  $w'_{j,2}$  without affecting the processing of other workloads. Since  $w_{i,1}$  finishes earlier than  $w_{j,1}$  with  $i < j$ , the swapping of the segments will not be constrained by their corresponding segments at the first processor. Moreover, the swapping of segments will not affect the total energy consumed, since they can still be processed at the same speed as before. Therefore, given a finishing order of  $w_{i,1}$  for all  $i$  at the first processor, the non-interleaving schedule of  $w_{i,2}$  will give a no worse result than any interleaving schedule.

Secondly, given the finishing order of  $w_{i,1}$  for all  $i$  at the first processor, for every  $w_{i,1}$ , the first processor has to complete  $\sum_{j=0}^i w_{j,1}$  in order to finish  $w_{i,1}$  because the finishing order is pre-defined. The non-interleaving schedule will make sure that every workload is finished as early as possible.

Therefore, changing an interleaving schedule to a non-interleaving one by the way shown above will result in no more energy consumption without delaying the finish time of any job.  $\square$

The energy minimization problem of 3 or more jobs is quite similar to that of 2 jobs as discussed in Section 3.1, except that we need to consider more precedence constraints, that is, we can only start processing  $w_{i,2}$  if processor 1 finishes processing  $w_{i,1}$ . In the following, we assume the execution order of the jobs are from  $J_1$  to  $J_n$ .

In the optimal schedule, if the finishing time of  $w_{i,1}$  is equal to the starting time of  $w_{i,2}$ , we say this time is a *critical time*. The time has “critical” properties

because it divides the problem into sub-problems, where jobs  $J_1$  to  $J_i$  can be analyzed separately from jobs  $J_{i+1}$  to  $J_n$ . It is easy to argue that workloads in the same segment must be processed using the same speed on the same processor because otherwise we can always slightly shift the workload to make the speed curve flatter without violating the precedence constraint between  $w_{i,2}$  and  $w_{i,1}$  (In the interior of a segment, the finishing time of  $w_{i,1}$  is strictly before the starting time of  $w_{i,2}$ ).

We further define job  $i$  to be a critical job if the finishing time of  $w_{i,1}$  is a critical time in the optimal schedule. Suppose that  $c_1, c_2, \dots, c_k$  are critical jobs (job  $J_n$  must be a critical job because we can always delay the finishing time of workload  $w_{n,1}$  to be equal to the starting time of  $w_{n,2}$  without increasing the energy consumption. Similarly job  $J_1$  is also a critical job). Then we can use the similar observation as the 2-job-2-processor case to calculate the corresponding minimum energy consumption to be

$$E_{min} = \frac{\left[ w_{1,1} + \sum_{i=1}^{k-1} \sqrt{\left( \sum_{j=c_i+1}^{c_{i+1}} w_{j,1} \right)^2 + \left( \sum_{j=c_i}^{c_{i+1}-1} w_{j,2} \right)^2} + w_{n,2} \right]^2}{d} \quad (1)$$

Hence, finding the optimal schedule for a given order of execution becomes finding critical jobs while guaranteeing workloads in the same segment can be executed continuously on two processors (with stable speed on each processor) without violating the constraint between  $w_{i,1}$  and  $w_{i,2}$ . This property is tested in the inequality inside the for loop in Algorithm 1, a dynamic programming algorithm we design to calculate the optimal schedule by guessing critical jobs. The intuition behind this testing is that as long as the proportion of the workloads of the type  $w_{i,1}$  till any possible critical time in the middle is less than the proportion of the workloads of type  $w_{i,2}$  in the same period, then using a stable speed on the same machine will always finish  $w_{i,1}$  earlier than  $w_{i-1,2}$ , which removes the possibility of critical time in between.

**Theorem 1.** *Procedure  $FindOptWork(1, n)$  can compute the optimal schedule when the processing sequence is fixed in  $O(n^3)$  time.*

*Proof.* The correctness is already proved in the above discussion. About the time complexity, there are  $n^2$  procedures to be called and each procedure takes  $O(n)$  time to finish if we preprocess all the testing in the IF statement in  $O(n^3)$  time. Therefore, the overall complexity is  $O(n^3)$ .  $\square$

However, how to find the optimal sequence remains an open question.

### 3.3 Optimal Sequencing for Jobs with the Same Workload on the Second Processor

In this subsection, we derive the optimal sequence for a line network of 2 processors with  $n$  jobs, where jobs have the same workload on the second processor. We

---

Procedure *FindOptWork*(*start, end*)

**Input:** *start* and *end*

**Output:** Optimal equivalent workload during this period

```

Min =  $\infty$ 
if start == end then
    return 0
end if
for i = start + 1 to end do
    /* testing whether start to i can be a segment in the optimal schedule
    if  $\frac{\sum_{j=\text{start}+1}^k w_{j,1}}{\sum_{j=\text{start}+1}^i w_{j,1}} \geq \frac{\sum_{j=\text{start}}^{k-1} w_{j,2}}{\sum_{j=\text{start}}^{i-1} w_{j,2}}$  for all  $\text{start} + 1 \leq k \leq i$  then
        Min =  $\min\{\text{Min}, \sqrt{(\sum_{j=\text{start}}^{k-1} w_{j,1})^2 + (\sum_{j=\text{start}}^{i-1} w_{j,2})^2}\}$  +
        FindOptWork(i, end)
    end if
end for
return Min

```

---

argue that the ascending order of workloads on the first processor can achieve optimality. We assume that the optimal sequence is  $J_1, J_2, \dots, J_n$ .

**Lemma 2.** *To achieve optimality,  $w_{1,1}$  should be the smallest among all  $w_{i,1}$ .*

**Lemma 3.** *To achieve optimality,  $w_{i,1} \leq w_{i+1,1}$  for  $i \in [2, n - 1]$ .*

**Theorem 2.** *The ascending order according to  $w_{i,1}$  is the optimal sequence.*

*Proof.* By Lemma 2 and Lemma 3, the ascending order of  $w_{i,1}$  is the optimal sequence to process jobs in the two processors.  $\square$

## 4 Star Network

In this section, we extend our analysis to a star network between  $s$  the sender and  $t$  the receiver as defined below:

**Definition 2.** *The star network consists of  $n + 1$  processors, out of which, the sender is connected to one intermediate processor  $v_1$ .  $v_1$  is then connected to  $n$  nodes  $v_2, v_3, \dots, v_{n+1}$ , all of which are connected to  $t$ . There will be exactly 1 job sent from the sender to the receiver via one of the nodes in  $\{v_i | 2 \leq i \leq n + 1\}$  during the time interval  $[0, d]$ . (We define  $n$  as the degree of the star)*

In such a structure, finding the optimal schedule is boiled down to finding an optimal sequence of scheduling on  $v_1$ . Hence, our focus in this section is to calculate the optimal sequence given different workload characteristics.

We assume that the jobs are executed in the order of  $J_1, J_2, \dots, J_n$ . For a star network, the possible schedule can be seen in Table 2.

Note that  $w_{i,2}$  shares the same processing time with  $w_{i+1,1}$  and  $w_{i+1,2}$  as a whole for  $1 \leq i < n$ . For the case of  $n = 2$ , the only difference with line networks

**Table 2.** Processing schedule in a star network

$v_1$	$v_2$	$v_3$	$\dots$	$v_{n+1}$
$w_{1,1}$	$w_{1,2}$			
$w_{2,1}$				
$\dots$				
$w_{n+1,1}$		$w_{2,2}$	$\vdots$	$w_{n+1,2}$

is that in the line networks,  $w_{1,2}$  shares the processing time with  $w_{2,1}$  only and  $w_{2,2}$  is processed after both. Therefore, the minimal energy consumption for the star network with  $n = 2$  is exactly the same as the following scenario:

- There are 2 processors in the line network,
  - There are 2 jobs to be processed: the first one  $J_1^{line}$  has  $w_{1,1}^{line} = w_{1,1}$  and  $w_{1,2}^{line} = w_{1,2}$ , and the second one  $J_2^{line}$  has  $w_{2,1}^{line} = w_{2,1} + w_{2,2}$  and  $w_{2,2}^{line} = 0$ .

Therefore, the minimal energy consumption

$$E = \frac{\left[ w_{1,1} + \sqrt{w_{1,2}^2 + (w_{2,1} + w_{2,2})^2} \right]^2}{d}$$

Note that this energy consumption is further equal to the energy consumption in the situation where

- There are 2 processors in the line network, and
  - There is 1 job with workloads  $w_{1,1}$  and  $\sqrt{w_{1,2}^2 + (w_{2,1} + w_{2,2})^2}$  respectively on the 2 processors

Recursively, for  $n \geq 2$ , the minimal energy consumption can be derived as

$$E = \frac{\left\{ w_{1,1} + \sqrt{w_{1,2}^2 + \left[ w_{2,1} + \sqrt{w_{2,2}^2 + \left( w_{3,1} + \sqrt{w_{3,2}^2 + \dots} \right)^2} \right]^2} \right\}^2}{d} \quad (2)$$

where  $w_{i,1} + \sqrt{w_{i,2}^2 + (w_{i+1,1} + \sqrt{w_{i+1,2}^2 + \dots})^2}$  is called the equivalent work-load from  $J_i$  to  $J_n$ .

#### 4.1 Optimal Sequencing for Jobs with the Same Workload on the Second Processor

In this subsection, we consider jobs whose workloads on the second processor are all the same. We show the following lemma.

**Lemma 4.** *In a star network, if all  $w_{i,2}$  are the same (we use  $w_{x,2}$  to represent it) and  $w_{i,1}$  is sorted in increasing order, then in the optimal schedule,  $w_{i,1}$  should be done before  $w_{j,1}$  if  $i < j$  and the minimized energy consumption is*

$$E = \frac{\left\{ w_{1,1} + \sqrt{w_{x,2}^2 + \left[ w_{2,1} + \sqrt{w_{x,2}^2 + \left( w_{3,1} + \sqrt{w_{x,2}^2 + \dots} \right)^2} \right]^2} \right\}^2}{d}.$$

## 4.2 Optimal Sequencing for Equal-Workload Jobs

In this subsection, we consider jobs whose workloads on the first processor and the second processor are the same, but different jobs can have different workloads. In other words, we have  $w_{i,1} = w_{i,2}$  for all  $1 \leq i \leq n$ . In the following, we use  $w_k$  to represent either  $w_{k,1}$  or  $w_{k,2}$ . Suppose that the optimal sequence is  $J_1, J_2, \dots, J_n$ .

**Lemma 5.** *In an optimal job sequencing for a star network, we have  $w_1 \geq w_2$ .*

**Lemma 6.** *In an optimal job sequencing, for any two neighboring workloads  $w_i$  and  $w_{i+1}$ , we have  $w_i \geq w_{i+1}$ .*

**Theorem 3.** *The optimal job sequencing should be in monotonically decreasing order with respect to the workloads.*

*Proof.* This theorem directly follows by Lemma 5 and 6.  $\square$

## 5 Conclusion

In this paper, we study the energy minimization problem in a network of processors, specifically line networks and star networks. We show that non-interleaving can achieve optimality when the line network contains two processors and the energy optimization problem in fact becomes an interesting sequencing problem which can be solved optimally in some special cases. On the other hand, for star networks, we characterized the optimal schedule when the workloads of jobs on the second level are the same. We also derived the optimal sequence for jobs whose workloads on both levels are the same. Although we assume  $P(s) = s^\alpha$  with  $\alpha = 2$  for the proof of the above properties, most of the properties can be easily extended for any  $\alpha > 1$ . For example, Minkowski inequality needs to be used instead of Cauchy inequality in the proof of Lemma 3. How to deal with the other cases remains an open problem. In both types of networks, finding the optimal sequence of execution given arbitrary workloads is a big challenge.

## References

1. Albers, S.: Algorithms for Dynamic Speed Scaling. In: STACS 2011, pp. 1–11 (2011)
2. Andrews, M., Fernandez, A., Zhang, L., Zhao, W.: Routing for Energy Minimization in the Speed Scaling Model. In: Proceedings of 29th IEEE International Conference on Computer Communications, pp. 2435–2443 (2010)
3. Bansal, N., Chan, H.-L., Lam, T.-W., Lee, L.-K.: Scheduling for speed bounded processors. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 409–420. Springer, Heidelberg (2008)
4. Bansal, N., Kimbrel, T., Pruhs, K.: Dynamic Speed Scaling to Manage Energy and Temperature. In: Proceedings of the 45th Annual Symposium on Foundations of Computer Science, pp. 520–529 (2004)
5. Chan, H.L., Chan, W.T., Lam, T.W., Lee, L.K., Mak, K.S., Wong, P.W.H.: Energy Efficient Online Deadline Scheduling. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 795–804 (2007)
6. Chan, J.W.-T., Lam, T.-W., Mak, K.-S., Wong, P.W.H.: Online deadline scheduling with bounded energy efficiency. In: Cai, J.-Y., Cooper, S.B., Zhu, H. (eds.) TAMC 2007. LNCS, vol. 4484, pp. 416–427. Springer, Heidelberg (2007)
7. Garey, M., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. *Math. of Oper. Res.* 1, 117–129 (1976)
8. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling theory: a survey. *Ann. Discrete Math.* 5, 287–326 (1979)
9. Hong, I., Qu, G., Potkonjak, M., Srivastavas, M.B.: Synthesis techniques for low-power hard real-time systems on variable voltage processors. In: Proceedings of the IEEE Real-Time Systems Symposium, pp. 178–187 (1998)
10. Irani, S., Pruhs, K.: Algorithmic Problems in Power Management. *ACM SIGACT News* 36(2), 63–76 (2005)
11. Ishihara, T., Yasuura, H.: Voltage Scheduling Problem for Dynamically Variable Voltage Processors. In: Proceedings of International Symposium on Low Power Electronics and Design, pp. 197–202 (1998)
12. Johnson, S.M.: Optimal Two- and Three-stage Production Schedules with Setup Times Included. *Naval Res. Logist. Quart.* 1, 61–68 (1954)
13. Kwon, W., Kim, T.: Optimal Voltage Allocation Techniques for Dynamically Variable Voltage Processors. In: Proceedings of the 40th Conference on Design Automation, pp. 125–130 (2003)
14. Lam, T.W., Lee, L.K., To, I.K.K., Wong, P.W.H.: Energy Efficient Deadline Scheduling in Two Processor Systems. In: Proceedings of the 18th International Symposium on Algorithm and Computation, pp. 476–487 (2007)
15. Li, M., Yao, F.F.: An Efficient Algorithm for Computing Optimal Discrete Voltage Schedules. *SIAM Journal on Computing* 35(3), 658–671 (2005)
16. Li, M., Yao, A.C., Yao, F.F.: Discrete and Continuous Min-Energy Schedules for Variable Voltage Processors. *Proceedings of the National Academy of Sciences USA* 103, 3983–3987 (2005)
17. Pinedo, M.: Scheduling: Theory, Algorithms, and Systems. In: Flow Shops and Flexible Flow Shops (Deterministic), 2nd edn., ch. 6, p. 129. Prentice-Hall, Englewood Cliffs (2002)
18. Pruhs, K., Stein, C.: How to Schedule When You Have to Buy Your Energy. In: Serna, M., Shaltiel, R., Jansen, K., Rolim, J. (eds.) APPROX and RANDOM 2010. LNCS, vol. 6302, pp. 352–365. Springer, Heidelberg (2010)

19. Wu, W., Li, M., Chen, E.: Min-Energy Scheduling for Aligned Jobs in Accelerate Model. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 462–472. Springer, Heidelberg (2009)
20. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science, pp. 374–382 (1995)

# Online Algorithms for Batch Machines Scheduling with Delivery Times\*

Peihai Liu and Xiwen Lu

Department of Mathematics, East China University of Science and Technology,

Shanghai, People's Republic of China, 200237

{pliu,xwlu}@ecust.edu.cn

**Abstract.** We consider online scheduling on  $m$  batch machines with delivery times. In this paper online means that jobs arrive over time and the characteristics of jobs are unknown until their arrival times. Once the processing of a job is completed it is delivered to the destination. The objective is to minimize the time by which all jobs have been delivered. For each job  $J_j$ , its processing time and delivery time are denoted by  $p_j$  and  $q_j$ , respectively. We first consider a restricted model: the jobs have agreeable processing and delivery times, i.e., for any two jobs  $J_i$  and  $J_j$ ,  $p_i > p_j$  implies  $q_i \geq q_j$ . For the restrict case, we provide a best possible online algorithm with competitive ratio  $1 + \alpha_m$ , where  $\alpha_m > 0$  is determined by  $\alpha_m^2 + m\alpha_m = 1$ . Then we present an online algorithm with a competitive ratio of  $1 + 2/\lfloor \sqrt{m} \rfloor$  for the general case.

**Keywords:** Scheduling, Online algorithm, Batch Machine, Delivery times.

## 1 Introduction

We consider an online scheduling model: online scheduling with delivery time on batch machines. Here, we have  $m$  batch machines and sufficiently many vehicles. There are  $n$  jobs  $J_1, J_2, \dots, J_n$ . Each job has a release time  $r_j$ , a processing time  $p_j$ , and a delivery time  $q_j$ . These characteristics about a job are known until it arrives. The objective is to minimize the time by which all jobs have been delivered. In this model, one batch machine can handle up to  $B$  jobs simultaneously as a batch, where  $B$  is sufficiently large. The processing time for a batch is equal to the longest processing time in the batch. All jobs in a common batch have the same starting time and completion time. Each job needs to be processed on one of the  $m$  batch machines, and once the job is completed we deliver it immediately to the destination by a vehicle. We denote by  $S_j$ ,  $C_j$  and  $L_j$ , respectively, the starting time of  $J_j$ , the completion time of  $J_j$  and the time by which  $J_j$  is delivered in a schedule. By using the general notation for a schedule problem, introduced

---

\* This work was supported by the National Nature Science Foundation of China(10771067,11101147) and the Fundamental Research Funds for the Central Universities.

by Graham et al.[3], this problem is denoted by  $Pm|online, r_j, q_j, B = \infty|L_{max}$ , where  $L_{max} = \max\{L_j : L_j = C_j + q_j, 1 \leq j \leq n\}$ .

Batch scheduling was first introduced by Lee et al.[6], which was motivated by burn-in operations in semiconductor manufacturing. Deng et al.[1] and Zhang et al.[16] studied the online scheduling problem on single batch machine. They proved that there is no online algorithm with competitive ratio smaller than  $(\sqrt{5} + 1)/2$ , and for the case  $B = \infty$ , they independently gave the same online algorithm with competitive ratio matching the lower bound. For the case  $B < n$ , the first online algorithm is a greedy heuristics, GRLPT, of Lee and Uzsoy[7] which was shown to be 2-competitive by Liu and Yu[9]. Zhang et al.[16] presented two online algorithms with competitive ratio not greater than 2. The above three online algorithms are all based on the ideas of the FBLPT rule. Later, Poon and Yu[11] presented a class of algorithms called the FBLPT-based algorithms that contains all the above three algorithms as special cases, and showed that any FBLPT-based algorithm has competitive ratio at most 2. In particular, for the case  $B = 2$ , they also gave an online algorithm with competitive ratio  $7/4$ .

For the online model of scheduling on  $m$  unbounded parallel batch machines, Zhang et al.[16] gave a lower bound  $\sqrt[m]{2}$ , and presented an online algorithm with competitive ratio  $1 + \alpha_m$ , where  $\alpha_m = (1 - \alpha_m)^{m-1}$ . For the special case where  $m = 2$ , Nong et al.[10] gave a  $\sqrt{2}$ -competitive online algorithm. For the general case, Liu et al.[8] and Tian et al.[13] showed the lower bound is  $1 + (\sqrt{m^2 + 4} - m)/2$ , and gave different optimal algorithms independently.

There have also been some results about online scheduling problems with delivery times. Hoogeveen and Vestjean[5] first studied the single-machine online scheduling problem with delivery times. They showed the lower bound is  $(\sqrt{5} + 1)/2$ , and gave a best possible deterministic online algorithm. On identical parallel machines, the lower bound couldn't be smaller than 1.5(See [14]). Hall and Shmoys[4] showed the competitive ratio of the LS algorithm is 2. Tian et al.[12] considered the single batch machine online scheduling problem with delivery times. They gave a 2-competitive algorithm for the unbounded case and a 3-competitive algorithm for the bounded case. In the same paper, they also studied a special model with identical processing times. They provided the best online algorithms with competitive ratio  $(\sqrt{5} + 1)/2$  for both bounded and unbounded cases. Yuan et al.[15] also studied the single machine parallel-batch scheduling. They provided a best possible online algorithm for two restricted models. Fang et al.[2] addressed the online scheduling on  $m$  unbounded batch machines with delivery times. They gave an online algorithm with competitive ratio  $1.5 + o(1)$ .

In this paper, we investigate online scheduling on  $m$  unbounded batch machines with delivery times. We first consider a restricted model: the jobs have agreeable processing and delivery times, i.e., for any two jobs  $J_i$  and  $J_j$ ,  $p_i > p_j$  implies  $q_i \geq q_j$ . We provide a best possible online algorithm with competitive ratio  $1 + \alpha_m$  for the problem, where  $\alpha_m^2 + m\alpha_m = 1$ . We also study the general case. We provide an online algorithm with a competitive ratio of  $r = 1 + 1/u + 1/v$ ,

where  $u, v$  are integers and  $uv \leq m$ . Especially when  $u = v = \lfloor \sqrt{m} \rfloor$  and  $m \rightarrow +\infty$ ,  $r \rightarrow 1$ .

Throughout this paper, we use  $r(S), p(S), q(S)$  to denote the smallest release time, the largest processing time and the largest delivery time of jobs in  $S$ , respectively.

## 2 A Lower Bound

We first consider the scheduling model  $Pm|online, r_j, B = \infty|C_{max}$ , which is a special case of the scheduling problems studied in this paper. Hence, its lower bound is also a lower bound of the problems we study. For the former, Liu et al.[8] and Tian et al.[13] presented the following lower bound of competitive ratio for all online algorithms independently.

**Lemma 1.** ([8,13] ) *There is no online algorithm with competitive ratio less than  $1 + \alpha_m$  for  $Pm|online, r_j, B = \infty|C_{max}$ , where  $\alpha_m^2 + m\alpha_m = 1$ .*

## 3 An Restrict Case

In this section, we deal with a restrict model: jobs have agreeable processing and delivery times, i.e., for any two jobs  $J_i$  and  $J_j$ ,  $p_i > p_j$  implies  $q_i \geq q_j$ . We provide a best possible online algorithm with competitive ratio  $1 + \alpha_m$  for the problem, where  $\alpha_m = (\sqrt{m^2 + 4} - m)/2$ , i.e.  $\alpha_m^2 + m\alpha_m = 1$ .

Let  $J(t)$  be the set of the jobs which are available but not yet scheduled at time  $t$ . Denote by  $p(t)$  the largest processing time of the jobs in  $J(t)$ . Denote by  $r(t)$  the smallest release time of the jobs in  $J(t)$ .

**Algorithm  $H_m^\infty(\alpha)$ :** At time  $t$ , if a machine is idle and there are available jobs but not yet scheduled and  $t \geq (1 + \alpha_m)r(t) + \alpha_m p(t)$ , then start all the available jobs as a single batch on the idle machine. Otherwise, do nothing but wait.

Now, we will prove that  $H_m^\infty(\alpha_m)$  has a competitive ratio of  $1 + \alpha_m$ . Let  $\sigma$  and  $\pi$  denote the schedule generated by the algorithm and an optimal off-line schedule, respectively. Their objective values are denoted by  $L_{max}(\sigma)$  and  $L_{max}(\pi)$ , respectively. We assume that there are  $b$  batches totally in  $\sigma$  which are written as  $B_1, B_2, \dots, B_b$ . For each  $i$ , the longest job in batch  $B_i$  is denoted by  $J_i$ (if two or more jobs have the longest processing time, in batch  $B_i$  let  $J_i$  be the job with the longest delivery time) with a release time  $r_i$ , a processing time  $p_i$  and a delivery time  $q_i$ . Since the jobs have agreeable processing and delivery times,  $p_i = p(B_i), q_i = q(B_i)$ . Thus we can use  $S_i(\sigma)$  and  $C_i(\sigma)$  to denote the starting time and the completion time of both job  $J_i$  and batch  $B_i$  in  $\sigma$ , respectively. It can be observed that  $S_i(\sigma) < S_j(\sigma)$  implies  $r(B_j) > S_i(\sigma)$  and  $S_j(\sigma) \geq (1 + \alpha_m)r(B_j) + \alpha_m p_j > (1 + \alpha_m)S_i(\sigma) + \alpha_m p_j$ . For any batch  $B_i$ , if  $S_i(\sigma) = (1 + \alpha_m)r(B_i) + \alpha_m p(B_i)$ , we say that  $B_i$  is regular.

For convenience, we assume that  $S_1(\sigma) < S_2(\sigma) < \dots < S_b(\sigma)$ .

**Lemma 2.** *In  $\sigma$ , if  $B_k$  is not a regular batch, then  $S_{k-1}(\sigma) > (1 - \alpha_m)S_k(\sigma)$ .*

*Proof.*  $B_k$  is not a regular batch means that  $S_k(\sigma) > (1 + \alpha_m)r(B_k) + \alpha_m p_k$ . Then by the algorithm, each machine is busy processing a batch during  $[r_k, S_k(\sigma))$ . Denote the  $m$  batches processed during  $[r_k, S_k(\sigma))$  by  $B_{k_1}, B_{k_2}, \dots, B_{k_m}$  with  $S_{k_1}(\sigma) < S_{k_2}(\sigma) < \dots < S_{k_m}(\sigma)$ . It is clear that  $k_m = k - 1$  and  $S_{k_j}(\sigma) + p_{k_j} \geq S_k(\sigma)$  ( $j = 1, 2, \dots, m$ ), i.e.,

$$p_{k_j} \geq S_k(\sigma) - S_{k_j}(\sigma), j = 1, 2, \dots, m \quad (1)$$

By the algorithm,  $S_{k_1}(\sigma) \geq \alpha_m p_{k_1} \geq \alpha_m(S_k(\sigma) - S_{k_1}(\sigma))$ . Thus

$$S_{k_1}(\sigma) \geq \frac{\alpha_m}{1 + \alpha_m} S_k(\sigma) \quad (2)$$

In addition, for each  $j$  ( $2 \leq j \leq m$ ),  $S_{k_j}(\sigma) \geq (1 + \alpha_m)r(B_{k_j}) + \alpha_m p(B_{k_j}) > (1 + \alpha_m)S_{k_{j-1}}(\sigma) + \alpha_m p_{k_j}$ . So using the inequality (1), we have,  $S_{k_j}(\sigma) \geq (1 + \alpha_m)S_{k_{j-1}}(\sigma) + \alpha_m(S_k(\sigma) - S_{k_j}(\sigma))$ , i.e.,

$$S_{k_j}(\sigma) > S_{k_{j-1}}(\sigma) + \frac{\alpha_m}{1 + \alpha_m} S_k(\sigma), 2 \leq j \leq m \quad (3)$$

Therefore,

$$S_{k_m}(\sigma) > \frac{m\alpha_m}{1 + \alpha_m} S_k(\sigma) = (1 - \alpha_m)S_k(\sigma) \quad (4)$$

i.e.  $S_{k-1}(\sigma) > (1 - \alpha_m)S_k(\sigma)$ .  $\square$

**Lemma 3.** *If  $B_k$  is not a regular batch, then  $p_k < \alpha_m S_k(\sigma)$ .*

*Proof.* If  $B_k$  is not a regular batch, then  $S_{k-1}(\sigma) > (1 - \alpha_m)S_k(\sigma)$  according to Lemma 2. By the algorithm, we know that  $S_k(\sigma) > (1 + \alpha_m)S_{k-1}(\sigma) + \alpha_m p_k$ . Thus,  $S_k(\sigma) > (1 - \alpha_m^2)S_k(\sigma) + \alpha_m p_k$  which implies that  $p_k < \alpha_m S_k(\sigma)$ .  $\square$

**Lemma 4.** *If  $B_k$  is not a regular batch, then each machine is busy processing a regular batch during  $[r_k, S_k(\sigma))$ .*

*Proof.*  $B_k$  is not a regular batch means that  $S_k(\sigma) > (1 + \alpha_m)r(B_k) + \alpha_m p_k$ . Then by the algorithm, each machine is busy processing a batch during  $[r_k, S_k(\sigma))$ . Denote the  $m$  batches processed in  $[r_k, S_k(\sigma))$  by  $B_{k_1}, B_{k_2}, \dots, B_{k_m}$ . Then

$$S_{k_j}(\sigma) + p_{k_j} \geq S_k(\sigma), j = 1, 2, \dots, m \quad (5)$$

Suppose for the sake of contradiction that  $B_{k_j}$  is not regular for some  $j$ . Then by Lemma 3,  $p_{k_j} < \alpha_m S_{k_j}(\sigma)$ . Thus  $S_k(\sigma) > (1 + \alpha_m)S_{k_j}(\sigma) > S_{k_j}(\sigma) + p_{k_j}$  which contradicts to the inequality (5). Therefore,  $B_{k_j}$  is regular for each  $j$  ( $1 \leq j \leq m$ ).

This completes the proof.  $\square$

**Theorem 1.**  $L_{max}(\sigma) \leq (1 + \alpha_m)L_{max}(\pi)$ .

*Proof.* Let  $B_l$  denote the first batch in  $\sigma$  that assumes the objective value  $L_{max}(\sigma)$ , i.e.,

$$L_{max}(\sigma) = S_l(\sigma) + p_l + q_l \quad (6)$$

If  $B_l$  is regular, then  $S_l(\sigma) = (1 + \alpha_m)r(B_l) + \alpha_m p_l$ . Thus  $L_{max}(\sigma) = (1 + \alpha_m)(r(B_l) + p_l) + q_l$ . It is clear that  $L_{max}(\pi) \geq r(B_l) + p_l + q_l$ . Therefore,  $L_{max}(\sigma) \leq (1 + \alpha_m)L_{max}(\pi)$ .

If  $B_l$  is not regular, then  $S_l(\sigma) > (1 + \alpha_m)r(B_l) + \alpha_m p_l$ . Thus by the algorithm, each machine is busy processing a batch during  $[r_l, S_l(\sigma))$ . Denote the  $m$  batches processed in  $[r_l, S_l(\sigma))$  by  $B_{l_1}, B_{l_2}, \dots, B_{l_m}$  with  $S_{l_1}(\sigma) < S_{l_2}(\sigma) < \dots < S_{l_m}(\sigma)$ . It is clear that  $l_m = l - 1$  and

$$S_l(\sigma) \leq S_{l_j}(\sigma) + p_{l_j}, j = 1, 2, \dots, m \quad (7)$$

According to Lemma 4,  $B_{l_j}$  is regular, i.e.

$$S_{l_j}(\sigma) = (1 + \alpha_m)r(B_{l_j}) + \alpha_m p_{l_j} \leq (1 + \alpha_m)r_{l_j} + \alpha_m p_{l_j} \quad (8)$$

Next, we will consider two cases according to the assignment of the  $m$  jobs  $J_{l_j}$  ( $1 \leq j \leq m$ ) in the optimal schedule  $\pi$ .

Case 1:  $S_{l_j}(\pi) < S_{l_j}(\sigma)$  for each  $j$  ( $1 \leq j \leq m$ ). Then  $r_{l_j} > S_{l_{j-1}}(\sigma) > S_{l_{j-1}}(\pi)$  for each  $j$  ( $2 \leq j \leq m$ ). Hence, the  $m$  jobs  $J_{l_j}$  ( $1 \leq j \leq m$ ) are processed in  $m$  different batches in  $\pi$ .

Using the equality (7), (8) and  $S_l(\sigma) > (1 + \alpha_m)S_{l_m}(\sigma)$ , we can obtain that  $C_{l_j}(\pi) \geq r_{l_j} + p_{l_j} \geq S_{l_m}(\sigma)$ . Recall that  $S_{l_j}(\pi) < S_{l_m}(\sigma)$  for each  $j$  ( $1 \leq j \leq m$ ). Hence the  $m$  jobs  $J_{l_j}$  ( $1 \leq j \leq m$ ) are grouped into  $m$  batches which are processed on  $m$  different machines in the optimal schedule  $\pi$ . Recall that  $r_l > S_{l_m}(\sigma) \geq S_{l_j}(\pi)$ . Thus, in the optimal schedule  $\pi$ ,  $J_l$  starts after some job  $J_{l_j}$ . So

$$L_{max}(\pi) \geq \min_{1 \leq j \leq m} \{r_{l_j} + p_{l_j}\} + p_l + q_l \quad (9)$$

While  $L_{max}(\sigma) \leq \min_j \{S_{l_j}(\sigma) + p_{l_j}\} + p_l + q_l \leq (1 + \alpha_m) \min_j \{r_{l_j} + p_{l_j}\} + p_l + q_l$ . Therefore,  $L_{max}(\sigma) \leq (1 + \alpha_m)L_{max}(\pi)$ .

Case 2:  $S_{l_j}(\pi) \geq S_{l_j}(\sigma)$  for some  $j$  ( $1 \leq j \leq m$ ). Then

$$L_{max}(\pi) \geq S_{l_j}(\sigma) + p_{l_j} \geq S_l(\sigma) \quad (10)$$

By Lemma 3, we have  $S_{l_m}(\sigma) > (1 - \alpha_m)S_l(\sigma)$ . Thus

$$L_{max}(\pi) \geq r_l + p_l + q_l > S_{l_m}(\sigma) + p_l + q_l \geq (1 - \alpha_m)S_l(\sigma) + p_l + q_l \quad (11)$$

Then it is from the equality  $\alpha_m \cdot (10) + (11)$  and (6) that  $L_{max}(\sigma) \leq (1 + \alpha_m)L_{max}(\pi)$ .

This completes the proof.  $\square$

## 4 The General Case

In this section, we present an online algorithm for the general case which has a competitive ratio of  $1 + 1/u + 1/v$ , where  $u, v$  are positive integers and  $uv \leq m$ .

Select  $uv$  machines and partition the  $uv$  machines into  $u$  sets:  $M^i (1 \leq i \leq u)$ . Each of  $M^i$  contains  $v$  machines. For convenience, we denote by  $M_j^i (1 \leq j \leq v)$  the machines in  $M^i (i = 1, 2, \dots, u)$ . If all machines in  $M^i$  are available, we call  $M^i$  available.

Let  $\alpha = 1/u, \beta = 1/v$ . Denote by  $A(t)$  the set containing all jobs that have arrived at or before time  $t$  and that have not been started by time  $t$ . We partition  $A(t)$  into  $v$  sets:  $A_i(t) = \{J_j | (i-1)\beta p(A(t)) \leq p_j < i\beta p(A(t)), J_j \in A(t)\}$ ,  $i = 1, 2, \dots, v-1$  and  $A_v(t) = \{J_j | p_j \geq (v-1)\beta p(A(t)), J_j \in A(t)\}$ .

**Algorithm  $H(u, v)$ :** Whenever  $M^i (i = 1, 2, \dots, l)$  is available and  $A(t) \neq \emptyset$ , make decision as follows. If  $t \geq (1+\alpha)r(A(t)) + \alpha p(A(t))$ , start  $A_j(t)$  as a batch on  $M_j^i$  for each  $j (1 \leq j \leq v)$ . Otherwise, wait.

For convenience, denote by  $B^x$  the set of the  $v$  batches with the same starting time  $t$ . Each of the  $v$  batches is denoted by  $B_y^x$  and  $B_y^x$  consists of jobs in  $A_y(t)$ . We call  $B^x$  a batch group.

Denote by  $\sigma$  the schedule produced by  $H(u, v)$ . For convenience, denote by  $S(B^i), C(B^i)$  the starting time and the maximum completion time of the batches in  $B^i$ . If  $S(B^i) = (1+\alpha)r(B^i) + \alpha p(B^i)$ , we say that  $B^i$  is regular.

**Lemma 5.** *In  $\sigma$ , any batch group  $B^i$  is regular, i.e.  $S(B^i) = (1+\alpha)r(B^i) + \alpha p(B^i)$ .*

*Proof.* Suppose to the contrary that  $B^i$  is not regular, i.e.  $S(B^i) > (1+\alpha)r(B^i) + \alpha p(B^i)$ . Thus between  $(1+\alpha)r(B^i) + \alpha p(B^i)$  and  $S(B^i)$ , there is a batch group which is processed on each machine set  $M^j (1 \leq j \leq u)$ . Suppose the  $u$  batch groups are  $B^{i_j} (1 \leq j \leq u)$  with  $S(B^{i_1}) < S(B^{i_2}) < \dots < S(B^{i_u})$ . Thus, for each  $j (1 \leq j \leq u)$ ,  $S(B^{i_j}) + p(B^{i_j}) \geq S(B^i)$ . Hence,

$$S(B^i) \leq \frac{1}{u} \sum_{j=1}^u (S(B^{i_j}) + p(B^{i_j})) = \alpha \sum_{j=1}^u C(B^{i_j}) \quad (12)$$

By the algorithm,

$$S(B^{i_1}) \geq \alpha p(B^{i_1}) \quad (13)$$

$$\begin{aligned} S(B^{i_j}) &\geq (1+\alpha)r(B^{i_j}) + \alpha p(B^{i_j}) \\ &> (1+\alpha)S(B^{i_{j-1}}) + \alpha p(B^{i_j}) \end{aligned} \quad (14)$$

Thus

$$\sum_{j=1}^u S(B^{i_j}) \geq (1+\alpha) \sum_{j=1}^{u-1} S(B^{i_j}) + \alpha \sum_{j=1}^u p(B^{i_j})$$

i.e.

$$S(B^{i_u}) \geq \alpha \sum_{j=1}^{u-1} S(B^{i_j}) + \alpha \sum_{j=1}^u p(B^{i_j})$$

Recall that  $S(B^i) > (1 + \alpha)S(B^{i_u})$ . We have that

$$\begin{aligned} S(B^i) &> S(B^{i_u}) + \alpha S(B^{i_u}) \\ &\geq \alpha \sum_{j=1}^{u-1} S(B^{i_j}) + \alpha \sum_{j=1}^u p(B^{i_j}) + \alpha S(B^{i_u}) \\ &= \alpha \sum_{j=1}^u (S(B^{i_j}) + p(B^{i_j})) \end{aligned} \tag{15}$$

which contradicts to the equality (12).

Therefore, in  $\sigma$ , any batch group  $B^i$  is regular, i.e.  $S(B^i) = (1 + \alpha)r(B^i) + \alpha p(B^i)$ .  $\square$

**Theorem 2.** *The competitive ratio of Algorithm  $H(u, v)$  is  $1 + \alpha + \beta$ .*

*Proof.* Let  $\pi$  be an optimal schedule.

In  $\sigma$ , let  $J_k$  be the first job that assumes the objective value  $L_{\max}(\sigma) = L_k(\sigma)$ . Suppose that  $J_k$  belongs to batch  $B_i^l$ . Since  $B^l$  is regular and  $p(B_i^l) \leq i\beta p(B^l)$ ,

$$L_{\max}(\sigma) = S(B^l) + p(B_i^l) + q_k \leq (1 + \alpha)r(B^l) + \alpha p(B^l) + i\beta p(B^l) + q_k \tag{16}$$

While in the optimal schedule  $\pi$ ,

$$L_{\max}(\pi) \geq r(B^l) + p_k + q_k \geq r(B^l) + (i - 1)\beta p(B^l) + q_k \tag{17}$$

$$L_{\max}(\pi) \geq r(B^l) + p(B^l) \tag{18}$$

Thus it is from the inequality (17)+( $\alpha + \beta$ )·(18) and (16) that  $L_{\max}(\sigma) \leq (1 + \alpha + \beta)L_{\max}(\pi)$ .

Hence, Algorithm  $H(u, v)$  is  $(1 + \alpha + \beta)$ -competitive.

Now, we can present an instance to show that the bound is tight. Let  $m = uv \geq 2$ , where  $u, v$  are integers. Suppose there are two jobs:  $J_1(r_1 = 0, p_1 = 1, q_1 = 0), J_2(r_2 = 0, p_2 = 1 - 1/v, q_2 = 1/v)$ . The algorithm will schedule  $J_1$  and  $J_2$  as a single batch which starts at  $1/u$ . Thus the object value will be  $1 + 1/u + 1/v$ . While the optimal schedule will assign  $J_1$  and  $J_2$  in different batches with starting time 0. Then the optimal value is 1. Hence the bound is tight.

This completes the proof.  $\square$

## References

1. Deng, X.T., Poon, C.K., Zhang, Y.Z.: Approximation algorithms in batch processing. *Journal of Combinatorial Optimization* 7, 247–257 (2003)
2. Fang, Y., Lu, X., Liu, P.: Online batch scheduling on parallel machines with delivery times. *Theoretical Computer Science* 412, 5333–5339 (2011)
3. Graham, R.L., Lawer, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics* 5, 287–326 (1979)
4. Hall, L.A., Shmoys, D.B.: Approximation schemes for constrained scheduling problems. In: *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pp. 134–139 (1989)
5. Hoogeveen, J.A., Vestjean, A.P.A.: A best possible deterministic online algorithm for minimizing maximum delivery times on a single machine. *SIAM Journal on Discrete Mathematics* 13, 56–63 (2000)
6. Lee, C.Y., Uzsoy, R., Martin-Vega, L.A.: Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research* 40, 764–775 (1992)
7. Lee, C.Y., Uzsoy, R.: Minimizing makespan on a single batch processing machine with dynamic job arrivals. *International Journal of Production Research* 37, 219–236 (1999)
8. Liu, P., Lu, X., Fang, Y.: A best possible deterministic online algorithm for minimizing makespan on parallel batch machines. *Journal of Scheduling* 15(1), 77–81 (2012)
9. Liu, Z.H., Yu, W.C.: Scheduling one batch processor subject to job release dates. *Discrete Applied Mathematics* 105, 129–136 (2000)
10. Nong, Q.Q., Cheng, T.C.E., Ng, C.T.: An improved on-line algorithm for scheduling on two unrestrictive parallel batch processing machines. *Operations Research Letters* 36, 584–588 (2008)
11. Poon, C.K., Yu, W.C.: On-line scheduling algorithms for a batch machine with finite capacity. *Journal of Combinatorial Optimization* 9, 167–186 (2005)
12. Tian, J., Fu, R., Yuan, J.: Online scheduling with delivery time on a single batch machine. *Theory Computer Science* 374, 49–57 (2007)
13. Tian, J., Cheng, T.C.E., Ng, C., Yuan, J.: Online scheduling on unbounded parallel-batch machines to minimize the makespan. *Information Processing Letters* 109, 1211–1215 (2009)
14. Vestjens, A.P.A.: Online machine scheduling. Ph.D. thesis, Department of mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands (1997)
15. Yuan, J., Li, S., Tian, J., Fu, R.: A best on-line algorithm for the single machine parallel-batch scheduling with restricted delivery times. *Jounral of Combinatorial Optimization* 17, 206–213 (2009)
16. Zhang, G., Cai, X., Wong, C.: On-line algorithms for minimizing makespan on batch processing machines. *Naval Research Logistics* 48, 241–258 (2001)

# How to Schedule the Marketing of Products with Negative Externalities\*

Zhigang Cao, Xujin Chen, and Changjun Wang

Academy of Mathematics and Systems Science  
Chinese Academy of Sciences, Beijing 100190, China  
[{zhigangcao,xchen,wcj}@amss.ac.cn](mailto:{zhigangcao,xchen,wcj}@amss.ac.cn)

**Abstract.** In marketing products with negative externalities, a schedule which specifies an order of consumer purchase decisions is crucial, since in the social network of consumers, the decision of each consumer is negatively affected by the choices of her neighbors. In this paper, we study the problems of finding a marketing schedule for two asymmetric products with negative externalities. The goals are two-fold: maximizing the sale of one product and ensuring regret-free purchase decisions. We show that the maximization is NP-hard, and provide efficient algorithms with satisfactory performance guarantees. Two of these algorithms give regret-proof schedules, i.e. they reach Nash equilibria where no consumers regret their previous decisions. Our work is the first attempt to address these marketing problems from an algorithmic point of view.

**Keywords:** Negative externality, Social network, Nash equilibrium, Efficient algorithm, Marketing schedule.

## 1 Introduction

The total value of any (consumer) product can be roughly classified into three parts: physical value, emotional value, and social value [12]. With the fast development of economy, the basic physical needs of more and more consumers are easily met. Consequently, people increasingly shift their attention to emotional and social values when they consider whether to buy a product. In particular, the social value, whose amount is not determined by what a consumer consumes alone or how she personally enjoys it, but by the comparisons with what other people around her consume, is becoming a more and more crucial ingredient for both consumer purchase and therefore seller marketing. For many products, whether they will be welcome depends mainly on how much social value they can provide to the consumers. This is especially true for fashionable and luxury goods, where the products often exhibit *negative (consumption) externalities* – they become less valuable as more people use them [1,10].

---

\* Supported in part by NNSF of China under Grant No. 11222109, 11021161, 10928102 and 71101140, 973 Project of China under Grant No. 2011CB80800 and 2010CB731405, and CAS under Grant No. kjcx-yw-s7.

The comparison that a consumer makes, for calculating the social value of a product, is naturally restricted to her neighbors in the social network. For a consumer, the social value of a product with negative externality is often proportional to the number of her neighbors who do not consume this product [10]. In the market, the purchase decisions of a consumer often depend on the values of the products at the time they are promoted – the product of larger value will be selected. In contrast to the physical and emotional values, which are relatively fixed, the social values of products vary with different marketing schedules. The goal of this paper is to design good marketing schedules for promoting products with negative externalities in social networks.

**Motivation and Related Work.** Our study is motivated by the practical marketing problem concerning how to bring the products to consumers' attention over time. Among a large literature on diffusion of competing products or opinions in social networks (see e.g., [2,8,9] and references therein), Chierichetti, Kleinberg and Pancones [8] recently studied the scheduling aspect of the diffusion problem on two products – finding an order of consumer purchase decision making to maximize the adoption of one product. In their model, the two competing products both have positive (consumption) externalities and every consumer follows the majority of her social network neighbors when the externalities outweigh her own internal preference. The authors [8] provided an algorithm that ensures an expected linear number of favorable decisions.

The network-related consumption externalities have been classified into four categories [7]. Comparing to the other three, the negative cross-consumer externality, as considered in this paper, has been far less studied [1,10], and was emphasized for its importance in management and marketing nowdays [7].

The model studied in this paper can also be taken as an extension of one side of *the fashion game*, which was formulated by Jackson [11]. Very interestingly, people often have quite different, in fact almost opposite, opinions on what is fashionable, e.g., “Lady Gaga is Goddess of fashion” vs “This year’s fashion color is black”. Following Jackson, we call consumers holding the former “personality reflection” idea of fashion *rebels* and the latter “prevailing style” idea *conformists*. More generally, a consumer behaves like a rebel (conformist) if the product, from her point of view, has negative (positive) externality. In an era emphasizing personal identities, more and more consumers would like to be rebels. For example, they would prefer Asian-style pants, when seeing many friends and colleagues (their social network neighbors) wearing European-style. However, the rebel social network is still under-researched in comparison with vast literature on conformist social networks. For a market where all the consumers are rebels, as considered in this paper, it has been previously studied by several papers under the term of anti-coordination [4,5].

**Model Formulation.** The market is represented by a social network  $G = (V, E)$ , an undirected graph with node set  $V$  consisting of  $n$  consumers and link set  $E$  of  $m$  connections between consumers. A seller has two (types of) products

$\mathcal{Y}$  and  $\mathcal{N}$  with similar functions. We abuse notations by using  $\mathcal{Y}$  and  $\mathcal{N}$  to denote both types and products.

The marketing is done sequentially: The seller is able to ask the consumers one by one whether they are more interested in  $\mathcal{Y}$  or in  $\mathcal{N}$ . Each consumer buys (chooses) exactly one of  $\mathcal{Y}$  and  $\mathcal{N}$ , whichever provides her a larger total value, only at the time she is asked. This is a simplification of the so called *precision marketing* [14]. For every consumer, a product of type  $\mathcal{T} \in \{\mathcal{Y}, \mathcal{N}\}$  provides her with total value  $p_{\mathcal{T}} + s_{\mathcal{T}}(x_{\mathcal{T}})$ , where  $p_{\mathcal{T}}$  is the sum of physical and emotional values, and  $s_{\mathcal{T}}(x_{\mathcal{T}})$  is the social value determined by decreasing function  $s_{\mathcal{T}}(\cdot)$  and the number  $x_{\mathcal{T}}$  of her neighbors who have bought product  $\mathcal{T}$ . We assume that  $\mathcal{Y}$  is very similar to  $\mathcal{N}$  with  $p_{\mathcal{Y}} > p_{\mathcal{N}}$  and the externality outweighs the physical and emotional difference, i.e., for any permutation  $\mathcal{T}, \mathcal{F}$  of  $\mathcal{Y}, \mathcal{N}$  and any nonnegative integers  $x, y$  ( $x < y$ ) we have  $s_{\mathcal{T}}(x) - s_{\mathcal{F}}(x) < p_{\mathcal{Y}} - p_{\mathcal{N}} < s_{\mathcal{T}}(x) - s_{\mathcal{F}}(y)$ .

Actually, the above model can be summarized as the following scheduling problems on rebel social networks.

*Rebels.* Every consumer is a rebel who, at her turn to choose from  $\{\mathcal{Y}, \mathcal{N}\}$ , will buy the product different from the one currently possessed by the majority of her neighbors. If there are equal numbers of neighbors having bought  $\mathcal{Y}$  and  $\mathcal{N}$  respectively, the consumer will always buy  $\mathcal{Y}$ .

*Scheduling.* A (marketing) *schedule*  $\pi$ , for network  $G$  is an ordering of consumers in  $V$  which specifies the order  $\pi(v) \in \{1, 2, \dots, n\}$  of consumer  $v \in V$  being asked to buy (choose)  $\mathcal{Y}$  or  $\mathcal{N}$ , or “being scheduled” for short. We refer to the problem of finding a schedule for a rebel social network as the *rebel scheduling problem*. Given schedule  $\pi$ , the choice (purchase *decision*) of each consumer  $v$  under  $\pi$  is uniquely determined, and we denote it by  $\pi[v]$ , which belongs to  $\{\mathcal{Y}, \mathcal{N}\}$ . The decisions of all consumers form the marketing *outcome*  $(\pi[v] : v \in V)$  of  $\pi$ . The basic goal of the rebel scheduling problem is to find a schedule whose outcome contains  $\mathcal{Y}$  (resp.  $\mathcal{N}$ ) decisions as many as possible because  $\mathcal{Y}$  (resp.  $\mathcal{N}$ ) is more profitable for the seller.

*Equilibrium.* As seen above, the value of a product changes as the marketing proceeds. Every schedule corresponds to a dynamic game among consumers. We assume that consumers behave naively without predictions. A natural question is: Can these simple behaviors (or equivalently, a schedule) eventually lead to a Nash equilibrium – a state where no consumer regrets her previous decision? This question is of both theoretical and practical interests. Schedules that lead to Nash equilibria are called *regret-proof*; they guarantee high consumer satisfaction, which is beneficial to the seller’s future marketing.

**Results and Contribution.** We prove that it is NP-hard to find a marketing schedule that maximizes the number of  $\mathcal{Y}$  (resp.  $\mathcal{N}$ ) decisions. Complementary to the NP-hardness, we design  $O(n^2)$ -time algorithms for finding schedules that guarantee at least  $n/2$  decisions of  $\mathcal{Y}$ , and at least  $n/3$  decisions of  $\mathcal{N}$ , respectively. The numbers  $n/2$  and  $n/3$  are best possible for any algorithm. Let  $\alpha$  denote the size of maximum independent set of  $G$ . We show that

regret-proof schedules that guarantee at least  $n/2$  decisions of  $\mathcal{Y}$  and at least  $\max\{\sqrt{n+1} - 1, (n - \alpha)/2\}$  decisions of  $\mathcal{N}$ , respectively, can be found in time  $O(mn^2)$ . In contrast, decentralized consumer choices without a schedule might result in an arbitrarily worse outcome. This can be seen from the star network, where in the worst case only one consumer chooses the product consistent with the seller's objective. The reader is referred to the full paper [6] for proofs and details omitted in the extended abstract.

To the best of our knowledge, this paper is the first attempt to address the scheduling problems for marketing products with negative externalities (i.e marketing in rebel social networks). Our algorithms for maximizing the number of  $\mathcal{Y}$  decisions can be extended to deal with the case of promoting one product where  $\mathcal{Y}$  and  $\mathcal{N}$  are interpreted as buying and not buying, respectively.

## 2 Maximization

We study the rebel scheduling problem to maximize seller's profits in Subsections 2.1 and 2.2, respectively, for the cases of  $\mathcal{Y}$  and  $\mathcal{N}$  having higher net profits.

Throughout we consider  $G = (V, E)$  a connected rebel social network for which we have  $n = O(m)$ . All results can be extended to any network without isolated nodes. Let  $\pi$  be a schedule for  $G$ , and  $u, v \in V$ . We say that  $\pi$  schedules  $v \in V$  with decision  $\pi[v] \in \{\mathcal{Y}, \mathcal{N}\}$ , and  $\pi$  schedules  $u$  before  $v$  if  $\pi(u) < \pi(v)$ .

### 2.1 When $\mathcal{Y}$ Is More Profitable

It is desirable to find an optimal schedule that maximizes the number of consumers purchasing  $\mathcal{Y}$ . Although this turns out to be a very hard task, we can guarantee that at least half of the consumers choose  $\mathcal{Y}$ .

**Theorem 1.** *The rebel scheduling problem for maximizing the number of  $\mathcal{Y}$  decisions is NP-hard.*

*Proof.* We prove by reduction from the *maximum independent set problem*. Given any instance of the maximum independent set problem on connected graph  $H = (N, F)$ , by adding some pendant nodes to  $H$  we construct in polynomial time a network  $G$  (an instance of the rebel scheduling problem): For each node  $u \in N$  with degree  $d(u)$  in  $H$ , we add a set  $P_u$  of  $d(u)$  nodes, and connect each of them to  $u$ . The resulting network  $G = (V, E)$  is specified by  $V := N \cup (\cup_{u \in N} P_u)$  and  $E := F \cup (\cup_{u \in N} \{up : p \in P_u\})$ , where each node in  $V \setminus N = \cup_{u \in N} P_u$  is *pendant*, and each node  $u \in N$  is *non-pendant* and has exactly  $2d(u)$  neighbors: half of them are non-pendant nodes in  $N$  and the other half are the  $d(u)$  pendant nodes in  $P_u$ .

We associate every schedule  $\pi$  for  $G$  with an integer  $\theta(\pi)$ , equal to the number of pendant nodes which are scheduled (by  $\pi$ ) after their unique neighbors. Clearly

$$\theta(\pi) \leq |V \setminus N| = 2|F| \text{ for any schedule } \pi \text{ of } G. \quad (2.1)$$

*Claim 1.* For any  $u \in N$  and any schedule  $\pi$  of  $G$ , if  $\pi$  schedules all nodes in  $P_u \cup \{u\}$  with  $\mathcal{Y}$ , then (all the  $d(u)$  pendant neighbors of  $u$  in  $P_u$  have to be scheduled before  $u$  with decisions  $\mathcal{Y}$ , therefore) all the  $d(u)$  non-pendant neighbors of  $u$  have to be scheduled with  $\mathcal{N}$  before  $u$  is scheduled.

Consider  $\pi$  being an optimal schedule for  $G$ . If  $\theta(\pi) = 0$ , then  $\pi$  schedules all pendant nodes before their neighbors, and hence all of these pendant nodes choose  $\mathcal{Y}$ . It follows from Claim 1 that  $\{v \in N : \pi[v] = \mathcal{Y}\}$  is an independent set of  $H$ . Since  $\pi$  is optimal, the independence set is maximum in  $H$ . Thus, in view of (2.1), to prove the theorem, it suffices to show the following.

*Claim 2.* Given an optimal schedule  $\pi$  for  $G$  with  $\theta(\pi) > 0$ , another optimal schedule  $\pi'$  for  $G$  with  $\theta(\pi') < \theta(\pi)$  can be found in polynomial time.

Since  $\theta(\pi) > 0$ , we can take  $w \in N$  to be the *last* non-pendant node scheduled by  $\pi$  earlier than some of its pendant neighbors. Under  $\pi$ , let  $P'_w$  ( $\emptyset \neq P'_w \subseteq P_w$ ) be the set of all pendant neighbors of  $w$  that are scheduled after  $w$ , let  $U$  be the set of non-pendant nodes scheduled after  $w$ , and let  $P_U$  be the set of the pendant nodes whose (non-pendant) neighbors belong to  $U$  (possibly  $U = \emptyset = P_U$ ). The choice of  $w$  implies that  $\pi$  schedules every node in  $P_U$  before its neighbor. Without loss of generality we may assume that under  $\pi$ ,

- (Pendant) nodes in  $P_U$  are scheduled before all other nodes (with  $\mathcal{Y}$ ).
- (Pendant) nodes in  $P'_w$  are scheduled immediately after  $w$  one by one.
- (Non-pendant) nodes in  $U$  are scheduled at last.

If  $\pi$  schedules  $w$  with  $\mathcal{N}$ , then at later time it schedules all pendant nodes in  $P'_w$  with  $\mathcal{Y}$ . Another optimal schedule  $\pi'$  (for  $G$ ) with the same outcome as  $\pi$  can be constructed as follows:  $\pi'$  schedules nodes in  $P'_w$  first, and then schedules other nodes of  $V$  in a relative order the same as  $\pi$ . Clearly,  $\pi'$  with  $\theta(\pi') \leq \theta(\pi) - 1$  is the desired schedule. It remains to consider the case where  $\pi$  schedules  $w$  with

$$\pi[w] = \mathcal{Y}. \quad (2.2)$$

It follows that  $\pi[p] = \mathcal{N}$  for all  $p \in P'_w$ . Let  $\pi'$  be the schedule that first schedules nodes of  $V \setminus \{w\}$  in a relative order the same as  $\pi$ , and schedules  $w$  finally. It is clear that  $\theta(\pi') \leq \theta(\pi) - 1$  and  $\pi'[p] = \mathcal{Y}$  for all  $p \in P'_w$ . We only need to show that  $\pi'$  is optimal.

Observe that  $\pi'$  first schedules every  $v \in V$  satisfying  $\pi(v) < \pi(w)$  with the same decision as in  $\pi$  (particularly, all nodes in  $P_U$  are scheduled with  $\mathcal{Y}$ ). Subsequently,  $\pi'$  schedules nodes in  $P'_w$  and  $U$  in the same relative order as  $\pi$ . Finally  $\pi'$  schedules  $w$ . Since all pendant nodes in  $P'_w$  ( $\neq \emptyset$ ) are scheduled by  $\pi'$  with  $\mathcal{Y}$ , and by  $\pi$  with  $\mathcal{N}$  the optimality of  $\pi'$  would follow if  $\pi'$  schedules every node of  $U$  with the same decision as  $\pi$ .

Suppose it were not the case. Let  $u \in U \subseteq N$  be the earliest node in  $U$  scheduled by  $\pi'$  with a decision  $\pi'[u]$  different from  $\pi[u]$ . It must be the case that  $w$  is a non-pendant neighbor of  $u$  and  $\pi[w] \neq \pi[u]$ . At the time  $\pi'$  schedules  $u$ , all pendant neighbors of  $u$  in  $P_u \subseteq P_U$  have been scheduled with  $\mathcal{Y}$  and the

non-pendant neighbor  $w$  has not been scheduled, it follows from Claim 1 that  $\pi'[u] = \mathcal{N}$ . As  $\pi[u] \neq \pi'[u]$  and  $\pi[w] \neq \pi[u]$ , we have  $\pi[w] = \pi'[u] = \mathcal{N}$ , a contradiction to (2.2). The optimality of  $\pi'$  is established, which proves Claim 2 and therefore Theorem 1.  $\square$

We next design an algorithm for finding a schedule that ensures at least  $n/2$  decisions of  $\mathcal{Y}$ , providing a 2-approximation to the optimal. The algorithm iteratively constructs a node set  $A$  for which there exist two schedules  $\pi'$  and  $\pi''$  scheduling each node in  $A$  with different decisions. In the end, at least half nodes of  $A$  can be scheduled by either  $\pi'$  or  $\pi''$  with  $\mathcal{Y}$  decisions. Subsequently, the nodes outside  $A$ , which form an independent set, will all choose  $\mathcal{Y}$  (in an arbitrary order).

*Algorithm 1.* Input: Network  $G = (V, E)$ . Output: Partial schedule  $\pi$  for  $G$ .

---

1. Initial setting:  $A \leftarrow \emptyset, t \leftarrow 1, \pi' \leftarrow$  a null schedule
  2. **While**  $\exists w \in V \setminus A$  which has different numbers of neighbors in  $A$  choosing  $\mathcal{Y}$  and  $\mathcal{N}$  respectively under  $\pi'$  **do**
  3.    SCHEDULE  $w$ :  $\pi'(w) \leftarrow t, \pi''(w) \leftarrow t$ ;  
 $A \leftarrow A \cup \{w\}, t \leftarrow t + 1$
  4. **End-while**
  5. **If**  $\exists uv \in E$  with  $u, v \notin A$   
**then** SCHEDULE  $uv$ :  $\pi'(u) \leftarrow t, \pi'(v) \leftarrow t + 1, \pi''(v) \leftarrow t, \pi''(u) \leftarrow t + 1$ ;  
 $A \leftarrow A \cup \{u, v\}, t \leftarrow t + 2$ ;  
Go back to Step 2.
  6. Let  $\pi$  be  $\pi'$  or  $\pi''$  whichever schedules more nodes with  $\mathcal{Y}$  (break tie arbitrarily)
- 

For convenience, we reserve symbol “SCHEDULE” for the scheduling (constructing  $\pi$  and  $\pi''$ ) as conducted at Steps 3 and 5 in Algorithm 1. Similarly, we also say “SCHEDULE a node” and “SCHEDULE an edge” with the implicit understanding that the node and the edge satisfy the conditions in Step 2 and Step 5 of Algorithm 1.

*Claim 3.*  $\pi'[v] = \mathcal{Y}$  if and only if  $\pi''[v] = \mathcal{N}$  for all  $v \in A$ .

*Proof.* The algorithm enlarges  $A$  gradually at Steps 3 and 5, producing a sequence of node sets  $A_0 = \emptyset, A_1, \dots, A_\ell = A$ . We prove by induction on  $k$  that  $\pi'(v) = \mathcal{Y}$  if and only if  $\pi''(v) = \mathcal{N}$  for all  $v \in A_k$ ,  $k = 0, 1, \dots, \ell$ . The base case of  $k = 0$  is trivial.

Suppose that  $k \geq 1$  and the statement is true for  $A_{k-1}$ . In case of  $A_k$  being produced at Step 2, suppose  $w$  has  $n_1$  (resp.  $n_2$ ) neighbors in  $A_{k-1}$  choosing  $\mathcal{Y}$  (resp.  $\mathcal{N}$ ) under  $\pi'$ . By hypothesis,  $w$  has  $n_1$  (resp.  $n_2$ ) neighbors in  $A_{k-1}$  choosing  $\mathcal{N}$  (resp.  $\mathcal{Y}$ ) under  $\pi''$ . Since  $n_1 \neq n_2$ , we see that  $\pi'[w] = \mathcal{Y}$  if and only if  $\pi''[w] = \mathcal{N}$ . In case of  $A_k$  being produced at Step 5, both  $u$  and  $v$  have equal number of neighbors in  $A_{k-1}$  choosing  $\mathcal{Y}$  and  $\mathcal{N}$ , respectively, under  $\pi'$ , due to the implementation of the while-loop at Steps 2–4. By hypothesis both  $u$  and  $v$  have equal number of neighbors in  $A_{k-1}$  choosing  $\mathcal{Y}$  and  $\mathcal{N}$ , respectively, under  $\pi''$ . It follows from  $uv \in E$  that  $\pi'[u] = \pi''[v] = \mathcal{Y}$  and  $\pi'[v] = \pi''[u] = \mathcal{N}$ . In either case, the statement is true for  $A_k$ , proving the claim.  $\square$

- Claim 4.* (i) At least half nodes of  $A$  are scheduled by  $\pi$  with  $\mathcal{Y}$  (by Step 6).  
(ii) The nodes in  $V \setminus A$  (if any) form an independent set of  $G$  (by Step 5).  
(iii) Each node in  $V \setminus A$  has an equal number of neighbors in  $A$  choosing  $Y$  and  $\mathcal{N}$ , respectively, under  $\pi'$  (by Steps 2–4), and under  $\pi''$  (by Claim 3), and hence under  $\pi$  (by Step 6).  $\square$

**Theorem 2.** *A schedule that ensures at least  $n/2$  decisions of  $\mathcal{Y}$  can be found in  $O(n^2)$  time.*

*Proof.* It follows from Claim 4(ii) and (iii) that  $\pi$  can be extended to a schedule for  $G$  such that all node in  $V \setminus A$  choose  $\mathcal{Y}$ . By Claim 4(i), the outcome contains at least  $n/2$  decisions of  $\mathcal{Y}$ .

Next we show the time complexity. Algorithm 1 keeps an  $n \times 2$  array  $[\delta_v, \delta'_v]$ ,  $v \in V$ , where  $\delta_v$  represents the difference between the numbers of neighbors of node  $v$  in  $A$  choosing  $\mathcal{Y}$  and  $\mathcal{N}$ , and  $\delta'_v$  represents the number of neighbors of node  $v$  in  $V \setminus A$ . The initial setting of the array  $[\delta_v, \delta'_v] = [0, \text{the degree of } v \text{ in } G]$ ,  $v \in V$ , takes  $O(n^2)$  time. Step 1 is to find a node  $w \notin A$  with  $\delta_w \neq 0$  by visiting  $\delta_v$ ,  $v \in V \setminus A$ . Step 5 is to find a node  $u \in V \setminus A$  with  $\delta'_u \geq 1$  and then find a neighbor  $u \in V \setminus A$  of  $v$ . The search in both Steps 2 and 5 takes  $O(n)$  time. Each time Algorithm 1 adds a node  $v$  to  $A$ , the algorithm updates the entries of  $v$ 's neighbors in the array, which takes  $O(n)$  time. Since we can add at most  $n$  nodes to  $A$ , Algorithm 1 terminates in  $O(n^2)$  time.  $\square$

The tightness of  $n/2$  in the above theorem can be seen from the case where the network  $G$  is a complete graph.

## 2.2 When $\mathcal{N}$ Is More Profitable

By reduction from the *bounded occurrence MAX-2SAT* problem [13], we obtain the following NP-hardness.

**Theorem 3.** *The rebel scheduling problem for maximizing the number of  $\mathcal{N}$  decisions is NP-hard.*  $\square$

Next, we design a 3-approximation algorithm for finding in  $O(n^2)$  time a schedule which ensures at least  $n/3$  decisions of  $\mathcal{N}$ . This is accomplished by a refinement of Algorithm 1 with some preprocessing.

The following terminologies will be used in our discussion. Given a graph  $H$  with node set  $U$ , let  $R, S \subseteq U$  be two node subsets. We say that  $R$  dominates  $S$  if every node in  $S$  has at least a neighbor in  $R$ . We use  $H \setminus R$  to denote the graph obtained from  $H$  by deleting all nodes in  $R$  (as well as their incident links). Thus  $H \setminus R$  is the subgraph of  $H$  induced by  $U \setminus R$ , which we also denote as  $H[U \setminus R]$ .

*Preprocessing.* Given a connected social network  $G = (V, E)$ , let  $X$  be any maximal independent set of  $G$ . It is clear that

- $X$  and  $Y := V \setminus X$  are disjoint node sets dominating each other.

We will partition  $X$  into  $X_1, \dots, X_\ell$  and  $Y$  into  $Y_0, Y_1, \dots, Y_\ell$  for some positive integer  $\ell$  such that Algorithm 1 schedules  $X_i \cup Y_i$  before  $X_{i-1} \cup Y_{i-1}$  for all  $i = \ell, \ell - 1, \dots, 2$ .

- Set  $G_0 = G$  and  $X_0 = \emptyset$ . Find  $Y_0 \subseteq Y$  such that  $Y \setminus Y_0$  is a *minimal* set that dominates  $X \setminus X_0 (= X)$  in graph  $G_0$ .
- Set graph  $G_1 := G \setminus (X_0 \cup Y_0) = G[(X \setminus X_0) \cup (Y \setminus Y_0)]$ .

The minimality of  $Y \setminus Y_0$  implies that in graph  $G_1$  every node in  $Y \setminus Y_0$  is adjacent to at least one pendant node in  $X \setminus X_0$ .

- Let  $X_1 \subseteq X \setminus X_0$  consist of all pendant nodes of  $G_1$  contained in  $X \setminus X_0$ .

If  $X \setminus (X_0 \cup X_1) \neq \emptyset$ , then  $Y \setminus Y_0$  still dominates  $X \setminus (X_0 \cup X_1)$ , and we repeat the above process with  $G_1, X \setminus X_0, Y \setminus Y_0$  in place of  $G_0, X, Y$ , respectively, and produce  $Y_1, G_2, X_2$  in place of  $Y_0, G_1, X_1$ .

Inductively, for  $i = 1, 2, \dots$ , given graph  $G_i = G \setminus \bigcup_{j=0}^{i-1} (X_j \cup Y_j) = G[(X \setminus \bigcup_{j=0}^{i-1} X_j) \cup (Y \setminus \bigcup_{j=0}^{i-1} Y_j)]$ , where  $Y \setminus \bigcup_{j=0}^{i-1} Y_j$  is a minimal set dominating  $X \setminus \bigcup_{j=1}^{i-1} X_j$ , and  $X_i$  the set of all pendant nodes of  $G_i$  contained in  $X \setminus \bigcup_{j=0}^{i-1} X_j$ , when  $X \setminus \bigcup_{j=0}^i X_j \neq \emptyset$ , we can

- Find  $Y_i \subseteq (Y \setminus \bigcup_{j=0}^{i-1} Y_j)$  such that  $Y \setminus \bigcup_{j=0}^i Y_j$  is a *minimal* set that dominates  $X \setminus \bigcup_{j=0}^i X_j$  in graph  $G_i$ .
- Set graph  $G_{i+1} := G \setminus \bigcup_{j=1}^i (X_j \cup Y_j) = G[(X \setminus \bigcup_{j=0}^i X_j) \cup (Y \setminus \bigcup_{j=0}^i Y_j)]$ .
- Let  $X_{i+1} \subseteq X \setminus \bigcup_{j=0}^i X_j$  consist of all pendant nodes of  $G_{i+1}$  that are contained in  $X \setminus \bigcup_{j=0}^i X_j$ .

The procedure terminates at  $i = \ell$  for which we have  $X \setminus \bigcup_{j=0}^\ell X_j = \emptyset$ , and

$$G_i = G[(\bigcup_{j=i}^\ell Y_j) \cup (\bigcup_{j=i}^\ell X_j)] \text{ for } i = 0, 1, \dots, \ell; \text{ in particular } G_0 = G.$$

Note that  $G_i \subseteq G_{i-1}$  for  $i = \ell, \ell - 1, \dots, 1$ ,  $Y \setminus Y_0$  is the disjoint union of  $Y_1, \dots, Y_\ell$ , and  $X$  is the disjoint union of  $X_1, \dots, X_\ell$ . The minimality of  $\bigcup_{j=i}^\ell Y_j = Y \setminus \bigcup_{j=0}^{i-1} Y_j$  implies that in graph  $G_i$  every node in  $\bigcup_{j=i}^\ell Y_j$  is adjacent to at least one pendant node in  $X_i$ .

*Claim 5.* For any  $i = \ell, \ell - 1, \dots, 1$ , in the subgraph  $G_i$ , all nodes in  $X_i$  are pendant, and every node in  $Y_i$  is adjacent to at least one node in  $X_i$ .

*Refinement.* Next we show that Algorithm 1 can be implemented in a way that all nodes of subgraph  $G_1$  are scheduled. If the implementation has led to at least  $n/3$  decisions of  $\mathcal{N}$ , we are done; otherwise, we can easily find another schedule that makes at least  $n/3$  nodes choose  $\mathcal{N}$ .

*Algorithm 2.* *Input:* Network  $G = (V, E)$  together with  $G_j, X_j, Y_j, j = 0, 1, \dots, \ell$ . *Output:* Partial schedule  $\pi$  for  $G$ .

- 
1. Initial setting:  $A \leftarrow \emptyset$
  2. **For**  $i = \ell$  **downto** 0 **do**
  3.   **While** in the subgraph  $G_i$ ,  $\exists w \in (X_i \cup Y_i) \setminus A$  which has different numbers of neighbors in  $A$  choosing  $\mathcal{Y}$  and  $\mathcal{N}$  respectively **do**

- 
4. SCHEDULE  $w$ ;  $A \leftarrow A \cup \{w\}$
  5. **End-while**
  6. **If**  $\exists$  edge  $uv$  of  $G_i$  with  $u, v \notin A$
  7.   **then** SCHEDULE  $uv$ ;  $A \leftarrow A \cup \{u, v\}$ ; Go back to Step 3.
  8. **End-for**
  9. Let  $\pi$  be a schedule for  $G[A]$  that schedules at least  $\frac{1}{2}|A|$  nodes with  $\mathcal{N}$
- 

The validity of Step 9 is guaranteed by Claim 3. Since  $X \cap Y_0 = \emptyset$ , the following claim implies  $X \subseteq A$ .

*Claim 6.*  $V \setminus A \subseteq Y_0$ .

*Proof.* We only need to show that each node  $w \in X_k \cup Y_k$  ( $k = 1, 2, \dots, \ell$ ) is selected to  $A$  when  $i = k$  in Algorithm 2.

In case of  $w \in X_k$ , it is pendant and has only one neighbor  $u$  in subgraph  $G_k$ . If  $u \in A$  when  $w$  is checked at Step 3, then  $w$  is selected to  $A$  at Step 4; Otherwise,  $w$  and  $u$  will be selected to  $A$  at the same time in Step 7.

In case of  $w \in Y_k$ , by Claim 5,  $w$  is adjacent to a pendant node  $v \in X_k$  of  $G_k$ . If, when checked at Step 3,  $w$  has different numbers of neighbors in  $A$  choosing  $\mathcal{Y}$  and  $\mathcal{N}$ , then it is selected to  $A$  at Step 4; otherwise, node  $v$  must have not been selected to  $A$ , and subsequently  $w$  and  $v$  are put into  $A$  together at Step 7.  $\square$

If  $|A| > 2n/3$ , then, by extending partial schedule  $\pi$  output by Algorithm 2, we obtain a schedule which makes at least  $n/3$  nodes choose  $\mathcal{N}$ . Otherwise,  $|V \setminus A| \geq n/3$ , and all nodes in  $V \setminus A$  can be scheduled with  $\mathcal{N}$  as follows: Schedule firstly the nodes in the maximal independent set  $X$  (all of them choose  $\mathcal{Y}$ ); secondly the nodes in  $V \setminus A$ , and finally all the other nodes. Recall that  $X$  dominates every node in  $Y \supseteq Y_0$ . It follows from Claim 6 that  $X$  dominates  $V \setminus A$ . As  $V \setminus A$  is an independent set in  $G$  (by Claim 4(ii)), the decisions of all nodes in  $V \setminus A$  are  $\mathcal{N}$ .

**Theorem 4.** *A schedule that ensures at least  $n/3$  decisions of  $\mathcal{N}$  can be found in  $O(n^2)$  time.*  $\square$

The tightness of  $n/3$  can be seen from a number of disjoint triangles linked by a path, where each triangle has exactly two nodes of degree two.

### 3 Regret-Proof Schedules

Using link cuts as a tool, we find regret-proof schedules that ensure at least  $n/2$  decisions of  $\mathcal{Y}$  and at least  $\sqrt{n+1} - 1$  decisions of  $\mathcal{N}$ , respectively.

Given  $G = (V, E)$ , let  $R$  and  $S$  be two disjoint subsets of  $V$ . We use  $[R, S]$  to denote the set of links (in  $E$ ) with one end in  $R$  and the other in  $S$ . If  $R \cup S = V$ , we call  $[R, S]$  a (link) *cut*. For a node  $v \in V$ , we use  $d_S(v)$  to denote the number of neighbors of  $v$  contained in  $S$ . *Each schedule  $\pi$  for  $G$  is associated with a cut  $[S_1, S_2]$  of  $G$  defined by its outcome:*  $S_1$  (resp.  $S_2$ ) is the

set of consumers scheduled with  $\mathcal{Y}$  (resp.  $\mathcal{N}$ ). A schedule  $\pi$  is *regret-proof* if and only if its associated cut  $[S_1, S_2]$  is *stable*, i.e., satisfies the following conditions:

$$d_{S_2}(v) \geq d_{S_1}(v) \text{ for any } v \in S_1, \text{ and } d_{S_1}(v) > d_{S_2}(v) \text{ for any } v \in S_2. \quad (3.1)$$

Note that  $S_1$  and  $S_2$  are asymmetric. For clarity, we call  $S_1$  the *leading set* of cut  $[S_1, S_2]$ . Any node that violates (3.1) is called *violating* (w.r.t.  $[S_1, S_2]$ ).

*Finding Stable Cuts.* A basic operation in our algorithms is “enlarging” unstable cuts by moving “violating” nodes from one side to the other. Let  $[S_1, S_2]$  be an unstable cut of  $G$  for which some  $v \in S_i$  ( $i = 1$  or  $2$ ) is violating. We define *type- $i$*  move of  $v$  (from  $S_i$  to  $S_{3-i}$ ) to be the setting:  $S_i \leftarrow S_i \setminus \{v\}$ ,  $S_{3-i} \leftarrow S_{3-i} \cup \{v\}$ , which changes the cut. The violation of (3.1) implies

- (a) type-1 move increases the cut size, and downsizes the leading set;
  - (b) type-2 move does not decrease the cut size, and enlarges the leading set.

Both types of moves are collectively called *moves*. Note that moves are only defined for violating nodes, and the cut size  $\|S_1, S_2\|$  is nondecreasing under moves. To find a stable cut, our algorithms work with a cut  $[S_1, S_2]$  of  $G$  and change it by moves sequentially. By (a) and (b), the number  $m_1$  of type-1 moves is  $O(m)$ . Moreover, we have the following observation.

**Lemma 1.** (i) From any given cut of size  $s$ ,  $O(m_1 + n)$  moves produce a stable cut (i.e., a cut without violating nodes) of size at least  $s + m_1$ .  
(ii) If the leading set of the stable cut produced is smaller than that of the given cut, then the number of type-2 moves is smaller than that of type-1 moves.

As a byproduct of (a) and (b), one can easily deduce that the rebel game on a network, where each rebel switches between two choices in favor of the minority choice of her neighbors, is a potential game and thus possesses a Nash equilibrium. The potential function is defined as the size of the cut between the rebels holding different choices.

**Lemma 2.** In  $O(n)$  time, either the current cut is verified to be stable, or a move is found and conducted.  $\square$

*Procedure 1. Input:* Cut  $[S_1, S_2]$  of  $G$ . *Output:* Stable cut  $[T_1, T_2] := \text{PRC1}(S_1, S_2)$



**Lemma 3.** Procedure 1 produces in  $O(tn + n^2)$  time a stable cut  $[T_1, T_2]$  of  $G$  such that  $|T_1| > n/2$ , where  $t = |[T_1, T_2]| - |[S_1, S_2]| \geq 0$ .

*Proof.* It follows from Lemma 1(i) that there are a number  $m'_1$  ( $\leq m$ ) of type-1 moves in total, and  $|[T_1, T_2]| \geq |[S_1, S_2]| + m'_1$ . By Lemma 2, it suffices to show that there are a total of  $O(m'_1 + n)$  moves.

Observe from Step 2 that each (implementation) of the while-loop at Step 3 starts with a cut whose leading set has at least  $n/2$  nodes. If this while-loop ends with a smaller leading set, by Lemma 1(ii) it must be the case that the while-loop conducts type-1 moves more times than conducting type-2 moves. Therefore after  $O(m'_1)$  moves, the procedure either terminates, or implements a while-loop that ends with a leading set  $S_1$  not smaller than one at the beginning of the while-loop. In the latter case, the until-condition at Step 4 is satisfied, and the procedure terminates. The number of moves conducted by the last while-loop is  $O(m'_1 + n)$  as implied by Lemma 1(i).  $\square$

*Regret-Proof Scheduling.* Our basic idea for finding regret-proof schedules goes as follows: Given a stable cut  $[S_1, S_2]$ , we try to schedule nodes in  $S_1$  (resp.  $S_2$ ) with  $\mathcal{Y}$  (resp.  $\mathcal{N}$ ) whenever possible. If not all nodes can be scheduled this way, we obtain another stable cut of larger size, from which we repeat the process.

*Algorithm 3.* *Input:* Cut  $[R_1, R_2]$  of network  $G$ . *Output:* A schedule for  $G$

- 
1. Initial setting:  $\mathcal{D}_1 \leftarrow \mathcal{Y}$ ,  $\mathcal{D}_2 \leftarrow \mathcal{N}$ ;  $T_i \leftarrow \emptyset$ ,  $S'_i \leftarrow R_i \setminus T_i$  ( $i = 1, 2$ )
  2. **Repeat**
  3.  $[S_1, S_2] \leftarrow \text{PRC1}(S'_1 \cup T_2, S'_2 \cup T_1)$   $// |[S_1, S_2]| \geq |[S'_1 \cup T_2, S'_2 \cup T_1]|$
  4. Set all nodes of  $G$  to be unscheduled
  5. **While**  $\exists$  unscheduled  $v \in S_i$  ( $i \in \{1, 2\}$ ) whose decision is  $\mathcal{D}_i$  **do** schedule  $v$
  6.  $T_i \leftarrow \{\text{scheduled nodes with decision } \mathcal{D}_i\}$ ,  $S'_i \leftarrow S_i \setminus T_i$  ( $i = 1, 2$ )  $// T_i \subseteq S_i$
  7. **Until**  $S'_1 = \emptyset$   $// \text{Until all nodes of } G \text{ are scheduled}$
  8. Output the final schedule for  $G$
- 

Note that cuts  $[S_1, S_2]$  returned by Procedure 1 at Step 3 are stable. At the end of Step 6, if  $S'_1 = \emptyset$ , then  $S'_2 = \emptyset$  (otherwise, every node  $v \in S'_2 \subseteq S_2$  satisfies  $d_{S_1}(v) = d_{T_1}(v) \leq d_{T_2}(v) \leq d_{S_2}(v)$ , saying that  $[S_1, S_2]$  is not stable.) Thus the condition in Step 7 is equivalent to saying “until all nodes of  $G$  are scheduled”.

**Theorem 5.** *Algorithm 3 finds in  $O(mn^2)$  time a regret-proof schedule with at least  $n/2$  decisions of  $\mathcal{Y}$ .*

*Proof.* Consider Step 6 setting  $S'_1 \neq \emptyset$ . Since nodes in  $S'_1 \cup S'_2$  cannot be scheduled, we have  $d_{T_1}(v) > d_{T_2}(v)$  for every  $v \in S'_1 = S_1 \setminus T_1$  and  $d_{T_2}(v) \geq d_{T_1}(v)$  for any  $v \in S'_2 = S_2 \setminus T_2$ , which gives

$$\begin{aligned} 0 &< \sum_{v \in S'_1} (d_{T_1}(v) - d_{T_2}(v)) + \sum_{v \in S'_2} (d_{T_2}(v) - d_{T_1}(v)) \\ &= (|[S'_1, T_1]| - |[S'_1, T_2]|) + (|[S'_2, T_2]| - |[S'_2, T_1]|) \\ &= |[S'_1 \cup T_2, S'_2 \cup T_1]| - |[S'_1 \cup T_1, S'_2 \cup T_2]|. \end{aligned}$$

Thus cut  $[S'_1 \cup T_2, S'_2 \cup T_1]$  has its size  $t > |[S'_1 \cup T_1, S'_2 \cup T_2]| = |[S_1, S_2]|$ . Subsequently, at Step 3, with input  $[S'_1 \cup T_2, S'_2 \cup T_1]$ , Procedure 1 returns a new

cut  $[S_1, S_2]$ , of size at least  $t$ , which is larger than the old one. It follows that the repeat-loop can only repeat a number  $k$  ( $\leq m$ ) of times.

By Lemma 3, for  $i = 1, 2, \dots, k$ , we assume that Procedure 1 in the  $i$ -th repetition (of Steps 3–6) returns in  $O(t_i n + n^2)$  time a cut whose size is  $t_i$  larger than the size of its input. Then  $\sum_{i=1}^k t_i \leq m$ , and overall Step 3 takes  $O(\sum_{i=1}^k (t_i n + n^2)) = O(mn^2)$  time. The overall running time follows from the fact that  $O(n^2)$  time is enough for finishing a whole while-loop at Step 5.

Note from Lemma 3 that the final cut  $[S_1, S_2]$  produced by Procedure 1 is stable and satisfies  $|S_1| \geq n/2$ . Since  $[S_1, S_2]$  is the cut associated with the final schedule output, the theorem is proved.  $\square$

By a similar but more complex algorithm, we can find in  $O(mn^2)$  time a regret-proof schedule that ensures at least  $\max\{\sqrt{n+1}-1, \frac{n-\alpha}{2}\}$  decisions of  $\mathcal{N}$ , where  $\alpha$  is the size of the maximum independent set of  $G$ .

## References

1. Adachi, T.: Third-degree price discrimination, consumption externalities and social welfare. *Economica* 72, 171–178 (2005)
2. Apt, K.R., Markakis, E.: Diffusion in social networks with competing products. In: Persiano, G. (ed.) SAGT 2011. LNCS, vol. 6982, pp. 212–223. Springer, Heidelberg (2011)
3. Borodin, A., Filmus, Y., Oren, J.: Threshold models for competitive influence in social networks. In: Saberi, A. (ed.) WINE 2010. LNCS, vol. 6484, pp. 539–550. Springer, Heidelberg (2010)
4. Bramoulle, Y.: Anti-coordination and social interactions. *Games and Economic Behavior* 58, 30–49 (2007)
5. Cao, Z., Yang, X.: A note on anti-coordination and social interactions. *Journal of Combinatorial Optimization* (2012) (online first), doi:10.1007/s10878-012-9486-7
6. Cao, Z., Chen, X., Wang, C.: How to schedule the marketing of products with negative externalities, CoRR abs/1303.6200 (2013)
7. Chiang, D.M., Teng, C.: Consumption externalities: review and future research opportunities. *Electroinic Commerce Studies* 3, 15–38 (2005)
8. Chierichetti, F., Kleinberg, J., Panconesi, A.: How to schedule a cascade in an arbitrary graph. In: Proc. 13th ACM Conference on Electronic Commerce, pp. 355–368 (2012)
9. Goyal, S., Kearns, M.: Competitive contagion in networks. In: Proc. 44th Symposium on Theory of Computing, pp. 759–774 (2012)
10. Holcombe, R.G., Sobel, R.S.: Consumption externalities and economic welfare. *Eastern Economic Journal* 26, 157–170 (2000)
11. Jackson, M.O.: Social and economic networks. Princeton University Press, Princeton (2008)
12. van Nes, N.: Understanding replacement behaviour and exploring design solutions. In: Cooper, T. (ed.) Longer Lasting Products: Alternatives to the Throwaway Society (2010)
13. Papadimitriou, C.H., Yannakakis, M.: Optimization, approximation, and complexity classes. *Journal of Computer and System Science* 43, 425–440 (1991)
14. Zabin, J., Brebach, G.: Precision Marketing: The New Rules for Attracting, Retaining and Leveraging Profitable Customers. John Wiley & Sons, Inc., Hoboken (2004)

# From Preemptive to Non-preemptive Speed-Scaling Scheduling

Evripidis Bampis<sup>1,\*</sup>, Alexander Kononov<sup>2,\*\*</sup>, Dimitrios Letsios<sup>1,3,\*</sup>, Giorgio Lucarelli<sup>1,3,\*</sup>, and Ioannis Neparis<sup>1,4</sup>

<sup>1</sup> LIP6, Université Pierre et Marie Curie, France

{Evripidis.Bampis, Giorgio.Lucarelli}@lip6.fr

<sup>2</sup> Sobolev Institute of Mathematics, Novosibirsk, Russia

alvenko@math.nsc.ru

<sup>3</sup> IBISC, Université d'Évry, France

dimitris.letsios@ibisc.univ-evry.fr

<sup>4</sup> Dept. of Informatics and Telecommunications, NKUA, Athens, Greece  
sdi0700181@di.uoa.gr

**Abstract.** We are given a set of jobs, each one specified by its release date, its deadline and its processing volume (work), and a single (or a set of) speed-scalable processor(s). We adopt the standard model in speed-scaling in which if a processor runs at speed  $s$  then the energy consumption is  $s^\alpha$  units of energy per time unit, where  $\alpha > 1$ . Our goal is to find a schedule respecting the release dates and the deadlines of the jobs so that the total energy consumption is minimized. While most previous works have studied the preemptive case of the problem, where a job may be interrupted and resumed later, we focus on the non-preemptive case where once a job starts its execution, it has to continue until its completion without any interruption. As the preemptive case is known to be polynomially solvable for both the single-processor and the multiprocessor case, we explore the idea of transforming an optimal preemptive schedule to a non-preemptive one. We prove that the preemptive optimal solution does not preserve enough of the structure of the non-preemptive optimal solution, and more precisely that the ratio between the energy consumption of an optimal non-preemptive schedule and the energy consumption of an optimal preemptive schedule can be very large even for the single-processor case. Then, we focus on some interesting families of instances: (i) equal-work jobs on a single-processor, and (ii) agreeable instances in the multiprocessor case. In both cases, we propose constant factor approximation algorithms. In the latter case, our algorithm improves the best known algorithm of the literature. Finally, we propose a (non-constant factor) approximation algorithm for general instances in the multiprocessor case.

---

\* Partially supported by the French Agency for Research under the DEFIS program TODO, ANR-09-EMER-010, by the project ALGONOW, co-financed by the European Union (European Social Fund - ESF) and Greek national funds, through the Operational Program “Education and Lifelong Learning”, under the program THALES, and by a French-Chinese Cai Yuanpei project.

\*\* Supported by the RFBR grant No 12-01-00184.

## 1 Introduction

One of the main mechanisms used for minimizing the energy consumption in computing systems and portable devices is the so called *speed-scaling mechanism* [1], where the speed of a processor may change dynamically. If the speed of the processor is  $s(t)$  at a time  $t$  then its power is  $s(t)^\alpha$ , where  $\alpha > 1$ , and the energy consumption is the power integrated over time. In this setting, we consider the *non-preemptive speed scaling* scheduling problem: we are given a set  $\mathcal{J}$  of  $n$  jobs, where each job  $J_j \in \mathcal{J}$  is characterized by its processing volume (work)  $w_j$ , its release date  $r_j$  and its deadline  $d_j$ , and a single (or a set of identical) speed-scalable processor(s). We seek for a *feasible*<sup>1</sup> *non-preemptive*<sup>2</sup> schedule of the jobs minimizing the overall energy consumption.

Recently, it has been proved that the non-preemptive speed scaling scheduling problem is  $\mathcal{NP}$ -hard even for the single-processor case [5]. Using the standard three-field notation we denote the single-processor (resp. multiprocessor) case as  $1|r_j, d_j|E$  (resp.  $P|r_j, d_j|E$ ). Our aim is to study the performance of one of the most standard approaches in scheduling for designing approximation algorithms: construct a non-preemptive schedule from an optimal preemptive one. We prove that for general instances this approach cannot lead to a constant factor approximation algorithm since the ratio between the energy consumption of an optimal non-preemptive schedule and the energy consumption of an optimal preemptive one can be very large even for the single-processor case. Despite this negative result, we show that for some important families of instances, this approach leads to interesting results. Moreover, we show how to use this approach in order to obtain an (non-constant) approximation algorithm for the general case.

### 1.1 Related Work

Different variants of the problem have been considered in the literature: with or without preemptions, with equal or arbitrary works, arbitrary release dates and deadlines or particular instances. The main families of instances, with respect to the release dates and the deadlines of the jobs, that have been studied are the following. In a *laminar instance*, for any two jobs  $J_j$  and  $J_{j'}$  with  $r_j \leq r_{j'}$  it holds that either  $d_j \geq d_{j'}$  or  $d_j \leq r_{j'}$ . In fact, such instances arise when recursive calls in a program create new jobs. In an *agreeable instance*, for any two jobs  $J_j$  and  $J_{j'}$  with  $r_j \leq r_{j'}$  it holds that  $d_j \leq d_{j'}$ , i.e. latter released jobs have latter deadlines. Such instances may arise in situations where the goal is to maintain a fair service guarantee for the waiting time of jobs. Note that agreeable instances correspond to proper interval graphs. In a *pure-laminar instance*, for any two jobs  $J_j$  and  $J_{j'}$  with  $r_j \leq r_{j'}$  it holds that  $d_j \geq d_{j'}$ . Note that the family of pure-laminar instances is a special case of laminar instances.

For the preemptive single-processor case  $(1|r_j, d_j, pmtn|E)$ , Yao et al. [16] proposed an optimal algorithm for finding a feasible schedule with minimum

---

<sup>1</sup> A schedule is *feasible* if every job is executed between its release date and its deadline.

<sup>2</sup> A schedule is *non-preemptive*, if every job is executed without interruption, i.e. once a job starts its execution it has to continue until its completion.

energy consumption. The multiprocessor case,  $P|r_j, d_j, pmtn|E$ , where there are  $m$  available processors has been solved optimally in polynomial time when both the preemption and the migration<sup>3</sup> of jobs are allowed [2,4,6,8].

Albers et al. [3] considered the multiprocessor problem where the preemption of the jobs is allowed but not their migration ( $P|r_j, d_j, pmtn, no-mig|E$ ). They first studied the problem where each job has unit work. They proved that it is polynomial time solvable for instances with agreeable deadlines. For general instances with unit-work jobs, they proved that the problem becomes strongly  $\mathcal{NP}$ -hard and they proposed an  $(\alpha^\alpha 2^{4\alpha})$ -approximation algorithm. For the case where the jobs have arbitrary works, the problem was proved to be  $\mathcal{NP}$ -hard even for instances with common release dates and common deadlines. Albers et al. proposed a  $2(2 - \frac{1}{m})^\alpha$ -approximation algorithm for instances with common release dates, or common deadlines, and an  $(\alpha^\alpha 2^{4\alpha})$ -approximation algorithm for instances with agreeable deadlines. Greiner et al. [12] gave a generic reduction transforming an optimal schedule for the multiprocessor problem with migration,  $P|r_j, d_j, pmtn|E$ , to a  $B_\alpha$ -approximate solution for the multiprocessor problem with preemptions but without migration,  $P|r_j, d_j, pmtn, no-mig|E$ , where  $B_\alpha$  is the  $\alpha$ -th Bell number. Note that the result of [12] holds only when  $\alpha \leq m$ .

Note that for the family of agreeable instances the assumption of preemption and no migration is equivalent to the non-preemptive assumption that we consider throughout this paper. For agreeable instances, any preemptive schedule can be transformed into a non-preemptive one of the same energy consumption, where the execution of each job  $J_j \in \mathcal{J}$  starts after the completion of any other job which is released before  $J_j$ . The correctness of this transformation can be proved by induction on the order where the jobs are released. Hence, the results of [3] and [12] for agreeable deadlines hold for the non-preemptive case as well.

Finally, Antoniadis and Huang [5] proved that the problem is  $\mathcal{NP}$ -hard even for pure-laminar instances. They also presented a  $2^{4\alpha-3}$ -approximation algorithm for laminar instances and a  $2^{5\alpha-4}$ -approximation algorithm for general instances. Notice that the polynomial-time algorithm for finding an optimal preemptive schedule presented in [16] returns a non-preemptive schedule when the input instance is agreeable.

In Table 1, we summarize the most related results of the literature. Several other results concerning scheduling problems in the speed-scaling setting have been presented, involving the optimization of some Quality of Service (QoS) criterion under a budget of energy, or the optimization of a linear combination of the energy consumption and some QoS criterion (see for example [7,9,15]). Moreover, two variants of the speed-scaling model considered in this paper have been studied in the literature, namely the *bounded speed model* in which the speeds of the processors are bounded above and below (see for example [10]), and the *discrete speed model* in which the speeds of the processors can be selected among a set of discrete speeds (see for example [14]). The interested reader can find more details in the recent survey [1].

---

<sup>3</sup> A schedule is migratory if a job may be interrupted and resumed on the same or on another processor. The parallel execution of parts of the same job is not allowed.

**Table 1.** Complexity and approximability results. (\*)The problem is equivalent with the corresponding non-preemptive problem.

Problem	Complexity	Approximation ratio
$1 r_j, d_j, pmtn E$	Polynomial [16]	—
$P r_j = 0, d_j = d, pmtn E$	Polynomial [11]	—
$P r_j, d_j, pmtn E$	Polynomial [2,4,6,8]	—
$P \text{agreeable}, w_j = 1, pmtn, no-mig E$ (*)	Polynomial [3]	—
$P r_j, d_j, w_j = 1, pmtn, no-mig E$	$\mathcal{NP}$ -hard ( $m \geq 2$ ) [3]	$B_\alpha$ [12]
$P r_j = 0, d_j = d, pmtn, no-mig E$ (*)	$\mathcal{NP}$ -hard [3]	PTAS [13,3]
$P r_j = 0, d_j, pmtn, no-mig E$ (*)	$\mathcal{NP}$ -hard	$\min\{2(2 - \frac{1}{m})^\alpha, B_\alpha\}$ [3,12]
$P \text{agreeable}, pmtn, no-mig} E$ (*)	$\mathcal{NP}$ -hard	$B_\alpha$ [12]
$P r_j, d_j, pmtn, no-mig E$	$\mathcal{NP}$ -hard	$B_\alpha$ [12]
$1 \text{agreeable} E$	Polynomial [16,5]	—
$1 \text{laminar} E$	$\mathcal{NP}$ -hard [5]	$2^{4\alpha-3}$ [5]
$1 r_j, d_j E$	$\mathcal{NP}$ -hard	$2^{5\alpha-4}$ [5]

## 1.2 Our Contribution

In this paper, we are interested in designing approximation algorithms for the non-preemptive speed-scaling scheduling problem using a standard approach in scheduling: given an optimal preemptive solution, design an algorithm to convert it into a feasible non-preemptive solution with as small degradation as possible in the approximation ratio. For the single-processor case, we use the optimal preemptive solution obtained by the algorithm of Yao et al. [16], while for the multiprocessor case, we use the preemptive migratory solution obtained by [2,4,6,8]. Unfortunately, the following proposition shows that for general instances the ratio between an optimal non-preemptive schedule to an optimal preemptive one can be very large even for the single-processor case. Due to space constraints we skip the proof and we will present it in the full version of this paper.

**Proposition 1.** *The ratio of the energy consumption of an optimal non-preemptive schedule to the energy consumption of an optimal preemptive schedule of the single-processor speed-scaling problem can be  $\Omega(n^{\alpha-1})$ .*

In what follows we show that for some particular instances, this approach leads to interesting results. In Section 3 we consider the single-processor case and we present an algorithm whose approximation ratio depends on the ratio of the maximum to the minimum work in the input instance. For equal-work jobs, our algorithm achieves an approximation ratio of  $2^\alpha$ . It has to be noticed here that the complexity status of this particular problem remains open and this is a challenging direction for future work. In Section 4 we consider the multiprocessor case. First, in Section 4.1, we deal with agreeable instances for which we present a  $(2 - \frac{1}{m})^{\alpha-1}$ -approximation algorithm. This ratio improves the best known approximation ratio of  $B_\alpha$  given in [12] for any  $\alpha > 1$ . For  $\alpha = 3$  our algorithm achieves a ratio of 4 while  $B_3 = 5$ , for  $\alpha = 4$  our algorithm achieves a ratio of 8 while  $B_4 = 15$ , etc. Note that in general  $B_\alpha = O(\alpha^\alpha)$ . Finally, in Section 4.2 we present an approximation algorithm of ratio  $m^\alpha (\sqrt[m]{n})^{\alpha-1}$  for general instances.

Before beginning, in the following section we present our notation and some preliminary known results that we use in our proofs.

## 2 Notation and Preliminaries

For the problems that we study in this paper, it is easy to see that in any optimal schedule, any job  $J_j \in \mathcal{J}$  runs at a constant speed  $s_j$  due to the convexity of the speed-to-power function. Given a schedule  $\mathcal{S}$  and a job  $J_j \in \mathcal{J}$ , we denote by  $E(\mathcal{S}, J_j) = w_j s_j^{\alpha-1}$  the energy consumed by the execution of  $J_j$  in  $\mathcal{S}$  and by  $E(\mathcal{S}) = \sum_{j=1}^n E(\mathcal{S}, J_j)$  the total energy consumed by  $\mathcal{S}$ . We denote by  $\mathcal{S}^*$  an optimal non-preemptive schedule for the input instance  $\mathcal{I}$ .

The following proposition has been proved in [5] for  $1|r_j, d_j|E$  but holds also for the corresponding problem on parallel processors.

**Proposition 2.** [5] *Suppose that the schedules  $\mathcal{S}$  and  $\mathcal{S}'$  process job  $J_j$  with speed  $s$  and  $s'$  respectively. Assume that  $s \leq \gamma s'$  for some  $\gamma \geq 1$ . Then  $E(\mathcal{S}, J_j) \leq \gamma^{\alpha-1} E(\mathcal{S}', J_j)$ .*

The following proposition has been proved in [3] and gives the relation between the energy consumption of an optimal single-processor preemptive schedule and an optimal multiprocessor preemptive schedule without migrations.

**Proposition 3.** [3] *For a given set of jobs  $\mathcal{J}$ , let  $\mathcal{S}$  be an optimal single-processor preemptive schedule and  $\mathcal{S}'$  be an optimal multiprocessor preemptive schedule without migrations. Then  $E(\mathcal{S}) \leq m^{\alpha-1} \cdot E(\mathcal{S}')$ .*

## 3 Single-Processor

In this section we consider the single-processor non-preemptive speed-scaling problem. We first prove some structural properties of the optimal preemptive schedule created by the algorithm in [16]. Then, we present an approximation algorithm for the non-preemptive case, using as lower bound the energy consumed by the optimal preemptive schedule. Our algorithm achieves a constant factor approximation ratio for equal-work jobs.

### 3.1 Properties of the Optimal Preemptive Schedule

We consider the time points  $t_0, t_1, \dots, t_k$ , in increasing order, where each  $t_\ell$ ,  $1 \leq \ell \leq k$ , corresponds to either a release date or a deadline, so that for each release date and deadline of a job there is a corresponding time point  $t_\ell$ . Then, we define the intervals  $I_{p,q} = [t_p, t_q]$ , for  $0 \leq p < q \leq k$ , and we denote by  $|I_{p,q}| = t_q - t_p$  the length of  $I_{p,q}$ . We say that a job  $J_j$  is alive in a given interval  $I_{p,q}$ , if  $[r_j, d_j] \subseteq I_{p,q}$ . The set of alive jobs in interval  $I_{p,q}$  is denoted by  $A(I_{p,q})$ . The density  $d(I_{p,q})$  of an interval  $I_{p,q}$  is the total work of its alive jobs over its length, i.e.,  $d(I_{p,q}) = \frac{\sum_{J_j \in A(I_{p,q})} w_j}{|I_{p,q}|}$ .

In [16], Yao et al. proposed a polynomial-time algorithm for finding an optimal schedule for  $1|r_j, d_j, pmtn|E$ . This algorithm schedules the jobs in distinct phases. More specifically, in each phase, the algorithm searches for the interval

of the highest density, denoted as  $I_{p,q}$ . All jobs in  $A(I_{p,q})$  are assigned the same speed, which is equal to the density  $d(I_{p,q})$ , and they are scheduled in  $I_{p,q}$  using the Earliest Deadline First (EDF) policy. We can assume, w.l.o.g., that in the case where two jobs have the same deadline, the algorithm schedules first the job of the smallest index. Then, the set of jobs  $A(I_{p,q})$  and the interval  $I_{p,q}$  are eliminated from the instance and the algorithm searches for the next interval of the highest density, and so on.

Given a preemptive schedule  $\mathcal{S}$  and a job  $J_j$ , let  $b_j(\mathcal{S})$  and  $c_j(\mathcal{S})$  be the starting and the completion time, respectively, of  $J_j$  in  $\mathcal{S}$ . For simplicity, we will use  $b_j$  and  $c_j$ , if the corresponding schedule is clear from the context. Note that there are no jobs with the same starting times, and hence all  $b_j$ 's are distinct. For the same reason, all  $c_j$ 's are distinct.

The following lemma describes some structural properties of the optimal schedule created by the algorithm in [16].

**Lemma 1.** *Consider the optimal preemptive schedule  $\mathcal{S}_{pr}$  created by the algorithm in [16]. For any two jobs  $J_j$  and  $J_{j'}$  in  $\mathcal{S}_{pr}$ , it holds that:*

- (i) *if  $b_j < b_{j'}$  then either  $c_j > c_{j'}$  or  $c_j \leq b_{j'}$ , and*
- (ii) *if  $b_j < b_{j'}$  and  $c_j > c_{j'}$  then  $s_j \leq s_{j'}$ .*

*Proof.*

(i) Assume for contradiction that there are two jobs  $J_j$  and  $J_{j'}$  in  $\mathcal{S}_{pr}$  with  $b_j < b_{j'}$ ,  $c_j < c_{j'}$  and  $c_j > b_{j'}$ .

We prove, first, that  $J_j$  and  $J_{j'}$  cannot be scheduled in a different phase of the algorithm. W.l.o.g., assume for contradiction that  $J_j$  is scheduled in a phase before  $J_{j'}$  and that  $I_{p,q}$  is the interval of the highest density in this phase. As  $b_j < b_{j'} < c_j$ , there is a subinterval  $I \subseteq [b_{j'}, c_j] \subset [b_j, c_j] \subseteq I_{p,q}$  during which  $J_{j'}$  is executed in  $\mathcal{S}_{pr}$ . By construction, each job is scheduled in a single phase and since  $I \subset I_{p,q}$ , it holds that  $[b_{j'}, c_{j'}] \subset I_{p,q}$ . Hence,  $J_j$  and  $J_{j'}$  are scheduled in the same phase.

The algorithm schedules  $J_j$  and  $J_{j'}$  using the EDF policy. Since the EDF policy schedules  $J_{j'}$  at time  $b_{j'}$  and  $b_{j'} < c_j$ , it holds that  $d_{j'} \leq d_j$ . In a similar way, since the EDF policy schedules  $J_j$  at time  $c_j$  and  $c_j < c_{j'}$ , it holds that  $d_j \leq d_{j'}$ . Hence,  $d_j = d_{j'}$ . However, since there is a tie, in both times  $b_{j'}$  and  $c_j$  the algorithm should have selected the same job, i.e., the job of the smallest index. Therefore, there is a contradiction on the way that algorithm works.

(ii) Assume for contradiction that there are two jobs  $J_j$  and  $J_{j'}$  in  $\mathcal{S}_{pr}$  with  $b_j < b_{j'}$ ,  $c_j > c_{j'}$  and  $s_j > s_{j'}$ . As  $s_j > s_{j'}$ ,  $J_j$  is scheduled in a phase before  $J_{j'}$ ; let  $I_{p,q}$  be the interval of the highest density in this phase. However, it holds that  $[b_{j'}, c_{j'}] \subset [b_j, c_j] \subseteq I_{p,q}$ , and hence  $J_{j'}$  should have been scheduled in the same phase as  $J_j$ , which is a contradiction.  $\square$

The above lemma implies that given an optimal preemptive schedule  $\mathcal{S}_{pr}$  obtained by the algorithm in [16], the interval graph, in which for each job  $J_j$  there is an interval  $[b_j, c_j]$ , has a laminar structure. Therefore, we can create a

tree-representation of  $\mathcal{S}_{pr}$  as follows. For each job  $J_j$  we create a vertex. For each pair of jobs  $J_j$  and  $J_{j'}$  with  $[b_{j'}, c_{j'}] \subset [b_j, c_j]$ , we create an arc  $(J_j, J_{j'})$  if and only if there is not a job  $J_{j''}$  with  $[b_{j''}, c_{j''}] \subset [b_{j'}, c_{j'}] \subset [b_j, c_j]$ . Note that, the created graph  $T = (V, E)$  is, in general, a forest. Moreover, using Lemma 1 we have that for each arc  $(J_j, J_{j'})$  it holds that  $s_j \leq s_{j'}$  in  $\mathcal{S}_{pr}$ . In other words, the speed of a job is at most equal to the speed of its children in  $T$ .

In what follows, we denote by  $T(J_j)$  the subtree of  $T$  rooted at vertex  $J_j \in V$ . Moreover, let  $n_j$  be the number of children of  $J_j$  in  $T$ .

**Lemma 2.** *Consider an optimal preemptive schedule  $\mathcal{S}_{pr}$  created by the algorithm in [16] and its corresponding graph  $T = (V, E)$ . Each job  $J_j$  is preempted at most  $n_j$  times in  $\mathcal{S}_{pr}$ .*

*Proof.* We will prove the lemma by induction on the tree.

Assume for contradiction that the root job  $J_r$  is preempted more than  $n_r$  times in  $\mathcal{S}_{pr}$ , that is the execution of  $J_r$  is partitioned into more than  $n_r + 1$  different maximal intervals. Thus, there is a child  $J_j$  of  $J_r$  and an interval  $I \subset [b_j, c_j]$  such that  $J_r$  is executed during  $I$ . Observe first that  $J_r$  and  $J_j$  should be scheduled in the same phase by the algorithm in [16]. Hence, the EDF policy is used and using similar arguments as in the proof of Lemma 1.(i) we have a contradiction.

For the induction step, assume for contradiction that the job  $J_j$  is preempted more than  $n_j$  times in  $\mathcal{S}_{pr}$ . Hence, either there is a child  $J_{j'}$  of  $J_j$  and an interval  $I \subset [b_{j'}, c_{j'}]$  such that  $J_j$  is executed during  $I$ , or there are two consecutive children  $J_{j'}$  and  $J_{j''}$  of  $J_j$  and two disjoint maximal intervals  $I$  and  $I'$ , with  $I, I' \subset [c_{j'}, b_{j''}]$ , such that  $J_j$  is executed during both  $I$  and  $I'$ . In the first case, we have a contradiction using similar arguments as for the base of the induction. In the second case, we get a contradiction using the inductive hypothesis.  $\square$

### 3.2 An Approximation Algorithm

In this section we present an approximation algorithm, whose ratio depends on  $w_{\max}$  and  $w_{\min}$ . In the case where all jobs have equal work to execute, this algorithm achieves a  $2^\alpha$ -approximation ratio. The main idea in Algorithm 1 is to transform the optimal preemptive schedule  $\mathcal{S}_{pr}$  created by the algorithm in [16] into a non-preemptive schedule  $\mathcal{S}_{npr}$ , based on the corresponding graph  $T = (V, E)$  of  $\mathcal{S}_{pr}$ . More specifically, the jobs are scheduled in three phases depending on the number (one, at least two or zero) of their children in  $T$ .

**Theorem 1.** *Algorithm 1 achieves an approximation ratio of  $(1 + \frac{w_{\max}}{w_{\min}})^\alpha$  for  $1|r_j, d_j|E$ .*

*Proof.* Consider first the jobs with exactly one child in  $T$ . By Lemma 2, each such job  $J_j$  is preempted at most once in  $\mathcal{S}_{pr}$ , and hence it is executed in at most two intervals in  $\mathcal{S}_{pr}$ . In  $\mathcal{S}_{npr}$  the whole work of  $J_j$  is scheduled in the largest of these two intervals. Thus, the speed of  $J_j$  in  $\mathcal{S}_{npr}$  is at most twice the speed of  $J_j$  in  $\mathcal{S}_{pr}$ . Therefore, using Proposition 2, for any job  $J_j$  with  $n_j = 1$  it holds that  $E(\mathcal{S}_{npr}, J_j) \leq 2^{\alpha-1} \cdot E(\mathcal{S}_{pr}, J_j)$ .

**Algorithm 1.**

- 
- 1: Create an optimal preemptive schedule  $\mathcal{S}_{pr}$  using the algorithm in [16];
  - 2: Create the corresponding graph  $T = (V, E)$  of  $\mathcal{S}_{pr}$ ;
  - 3: Create the non-preemptive schedule  $\mathcal{S}_{npr}$  as follows:
  - 4: **for** each job  $J_j$  with  $n_j = 1$  **do**
  - 5:   Schedule non-preemptively the whole work of  $J_j$  in the biggest interval where a part of  $J_j$  is executed in  $\mathcal{S}_{pr}$ ;
  - 6: **for** each remaining non-leaf job  $J_j$  **do**
  - 7:   Find an unlabeled leaf job  $J_{j'} \in T(J_j)$ ; Label  $J_{j'}$ ;
  - 8:   Schedule non-preemptively  $J_j$  and  $J_{j'}$  with the same speed in the interval where  $J_{j'}$  is executed in  $\mathcal{S}_{pr}$ ;
  - 9: Schedule the remaining leaf jobs as in  $\mathcal{S}_{pr}$ ;
  - 10: **return**  $\mathcal{S}_{npr}$ ;
- 

Consider now the remaining non-leaf jobs. As for each such job  $J_j$  it holds that  $n_j \geq 2$ , in the subtree  $T(J_j)$  the number of non-leaf jobs with  $n_j \geq 2$  is smaller than the number of leaf jobs. Hence, we can create an one-to-one assignment of the non-leaf jobs with  $n_j \geq 2$  to leaf jobs such that each non-leaf job  $J_j$  is assigned to an unlabeled leaf job  $J_{j'} \in T(J_j)$ .

Consider a non-leaf job  $J_j$  with  $n_j \geq 2$  and its assigned leaf job  $J_{j'} \in T(J_j)$ . Recall that leaf jobs are executed non-preemptively in  $\mathcal{S}_{pr}$ . Let  $I$  be the interval in which  $J_{j'}$  is executed in  $\mathcal{S}_{pr}$ . The speed of  $J_{j'}$  in  $\mathcal{S}_{pr}$  is  $s_{j'} = \frac{w_{j'}}{|I|}$  and its energy consumption is  $E(\mathcal{S}_{pr}, J_{j'}) = w_{j'} s_{j'}^{\alpha-1}$ . In  $\mathcal{S}_{npr}$  both  $J_j$  and  $J_{j'}$  are executed during  $I$  with speed  $s = \frac{w_j + w_{j'}}{|I|}$ . The energy consumed by  $J_j$  and  $J_{j'}$  in  $\mathcal{S}_{npr}$  is

$$\begin{aligned}
E(\mathcal{S}_{npr}, J_j) + E(\mathcal{S}_{npr}, J_{j'}) &= (w_j + w_{j'}) s^{\alpha-1} = (w_j + w_{j'}) \left( \frac{w_j + w_{j'}}{|I|} \right)^{\alpha-1} \\
&= (w_j + w_{j'})^\alpha \left( \frac{s_{j'}}{w_{j'}} \right)^{\alpha-1} = \left( \frac{w_j + w_{j'}}{w_{j'}} \right)^\alpha \cdot w_{j'} s_{j'}^{\alpha-1} \\
&= \left( \frac{w_j + w_{j'}}{w_{j'}} \right)^\alpha \cdot E(\mathcal{S}_{pr}, J_{j'}) \\
&< \left( \frac{w_{\max} + w_{\min}}{w_{\min}} \right)^\alpha \cdot (E(\mathcal{S}_{pr}, J_j) + E(\mathcal{S}_{pr}, J_{j'}))
\end{aligned}$$

Moreover, note that  $J_j$  is alive during  $I$  and hence  $\mathcal{S}_{npr}$  is a feasible schedule.

Finally, for each remaining leaf job  $J_j$ , it holds that  $E(\mathcal{S}_{npr}, J_j) = E(\mathcal{S}_{pr}, J_j)$ , concluding the proof of the theorem.  $\square$

When all jobs have equal work to execute, the following corollary holds.

**Corollary 1.** *Algorithm 1 achieves an approximation ratio of  $2^\alpha$  for  $1|w_j = w, r_j, d_j|E$ .*

Note that the complexity of  $1|w_j = w, r_j, d_j|E$  is not known and it is an interesting open question.

## 4 Parallel Processors

In this section, we show how to use the optimal preemptive schedule to achieve approximation algorithms for the multiprocessor case. We first present a constant factor approximation algorithm for instances with agreeable deadlines. Then, we consider general instances. As by Proposition 1 we know that the energy consumption of an optimal preemptive schedule can be  $\Omega(n^{\alpha-1})$  far from the energy consumption of an optimal non-preemptive schedule, we give an algorithm for the latter case that uses as a lower bound the optimal preemptive schedule and achieves an approximation factor that depends on  $n$  and  $m$ .

### 4.1 Agreeable Instances

The problem  $P|{\text{agreeable}}|E$  is known to be  $\mathcal{NP}$ -hard [3] and  $B_\alpha$ -approximable [12]. In this section we present an approximation algorithm of ratio  $(2 - \frac{1}{m})^{\alpha-1}$ , which is better than  $B_\alpha$  for any  $\alpha > 1$ .

Our algorithm creates first an optimal preemptive schedule, using one of the algorithms in [2,4,6,8]. The total execution time  $e_j$  of each job  $J_j \in \mathcal{J}$  in this preemptive schedule is used to define an appropriate processing time  $p_j$  for  $J_j$ . Then, the algorithm schedules non-preemptively the jobs using these processing times according to the Earliest Deadline First policy, i.e., at every time that a processor becomes idle, the non-scheduled job with the minimum deadline is scheduled on it. The choice of the values of the  $p_j$ 's has been made in such a way that the algorithm completes all the jobs before their deadlines.

---

#### Algorithm 2.

---

- 1: Create an optimal multiprocessor preemptive schedule  $\mathcal{S}_{pr}$ ;
  - 2: Let  $e_j$  be the total execution time of the job  $J_j \in \mathcal{J}$ , in  $\mathcal{S}_{pr}$ ;
  - 3: Schedule non-preemptively the jobs with the Earliest Deadline First (EDF) policy, using the appropriate speed such that the processing time of the job  $J_j \in \mathcal{J}$ , is equal to  $p_j = e_j / (2 - \frac{1}{m})$ , obtaining the non-preemptive schedule  $\mathcal{S}_{npr}$ ;
  - 4: **return**  $\mathcal{S}_{npr}$ ;
- 

**Theorem 2.** *Algorithm 2 achieves an approximation ratio of  $(2 - \frac{1}{m})^{\alpha-1}$  for  $P|{\text{agreeable}}|E$ .*

*Proof.* We consider the jobs indexed in non-decreasing order of their release dates/deadlines. In what follows, we denote by  $b_j$  the starting time of the job  $J_j \in \mathcal{J}$  in  $\mathcal{S}_{npr}$ . Hence, the completion time  $c_j$  of  $J_j$  in  $\mathcal{S}_{npr}$  is  $c_j = b_j + p_j$ . We first show that the  $\mathcal{S}_{npr}$  is a feasible schedule. In other words, we will prove that for the completion time of the job  $J_j \in \mathcal{J}$ , it holds that  $c_j \leq d_j$ . Before that we introduce some additional notation.

Note that at each time either all processors execute some job or there is at least one processor which is idle. Based on this observation, we partition  $\mathcal{S}_{npr}$  into

maximal intervals: the “full” and the “non-full” intervals. At each time during a “full” interval, every processor executes some job. At each time during a “non-full” interval, there is at least one processor which is idle. Let  $\ell$  be the number of the “non-full” intervals. Let  $[\tau_i, t_i]$ ,  $1 \leq i \leq \ell$ , be the  $i$ -th “non-full” interval. Hence,  $[t_{i-1}, \tau_i]$ ,  $1 \leq i \leq \ell + 1$ , is a “full” interval. For convenience,  $t_0 = 0$  and  $\tau_{\ell+1} = \max_{J_j \in \mathcal{J}} \{c_j\}$ . Note that the schedule can start at a “non-full” interval, i.e.,  $t_0 = \tau_1$ , or can end with a “non-full” interval, i.e.,  $t_\ell = \tau_{\ell+1}$ .

Consider first a job  $J_j \in \mathcal{J}$  that is released during a “non-full” interval  $[\tau_i, t_i]$ . Since the jobs are scheduled according to EDF policy,  $J_j$  starts its execution at its release date, i.e.,  $b_j = r_j$ . Given that  $J_j$  has smaller processing time in  $\mathcal{S}_{npr}$  than in  $\mathcal{S}_{pr}$  and as  $\mathcal{S}_{pr}$  is a feasible schedule, it holds that  $c_j \leq d_j$ .

Consider now a job  $J_j \in \mathcal{J}$  that is released during a “full” interval  $[t_i, \tau_{i+1}]$ . We denote by  $\mathcal{J}_i = \{J_j \in \mathcal{J} : r_j < t_i\}$  the set of jobs which are released before  $t_i$ . Let  $P_{npr,i}$  be the amount of time that the jobs in  $\mathcal{J}_i$  are executed after  $t_i$  in  $\mathcal{S}_{npr}$  and  $E_{pr,i}$  be the amount of time that the jobs in  $\mathcal{J}_i$  are executed after  $t_i$  in  $\mathcal{S}_{pr}$ . We need the following claim (due to space constraints we skip its proof).

*Claim.* For each  $i$ ,  $0 \leq i \leq \ell$ , it holds that  $P_{npr,i} \leq \frac{E_{pr,i}}{(2 - \frac{1}{m})}$ .

Let  $J_q$  be the first job which is released after  $t_i$ , i.e.,  $r_q = t_i$ . For  $J_j$  we have

$$c_j \leq t_i + \frac{P_{npr,i} + \sum_{k=q}^{j-1} p_k}{m} + p_j \leq t_i + \frac{\frac{E_{pr,i} + \sum_{k=q}^{j-1} e_k}{m} + e_j}{(2 - \frac{1}{m})}$$

As  $\mathcal{S}_{pr}$  is a feasible schedule, all jobs  $J_q, \dots, J_j$  are executed inside the interval  $[t_i, d_j]$  in  $\mathcal{S}_{pr}$  and at least  $E_{pr,i}$  amount of time of the jobs in  $\mathcal{J}_i$  are also executed in the same time interval, it holds that  $E_{pr,i} + \sum_{k=q}^j e_k \leq m(d_j - t_i)$  and  $e_j \leq d_j - t_i$ . Therefore, we obtain that  $c_j \leq t_i + (2 - \frac{1}{m}) \frac{d_j - t_i}{(2 - \frac{1}{m})} = d_j$ .

Finally, we have to prove the approximation ratio of our algorithm. When dividing the execution time of all jobs by  $(2 - \frac{1}{m})$ , at the same time the speed of each job is multiplied by the same factor. Using Proposition 2 we have that

$$E(\mathcal{S}_{npr}) \leq (2 - \frac{1}{m})^{\alpha-1} E(\mathcal{S}_{pr}) \leq (2 - \frac{1}{m})^{\alpha-1} E(\mathcal{S}^*)$$

since the energy consumed by the optimal preemptive schedule  $\mathcal{S}_{pr}$  is a lower bound to the energy consumed by an optimal non-preemptive schedule  $\mathcal{S}^*$  for the input instance  $\mathcal{I}$ .  $\square$

## 4.2 General Instances

In this section we present an approximation algorithm for the multiprocessor non-preemptive speed scaling problem  $P|r_j, d_j|E$ . The main idea of our algorithm is to create an optimal single-processor preemptive schedule for the set of jobs  $\mathcal{J}$ . The jobs which are preempted at most  $n^{\frac{1}{m}}$  times in this schedule, are

scheduled non-preemptively on processor 1. For the remaining jobs we create again an optimal single-processor preemptive schedule, we use processor 2 for the jobs which are preempted at most  $n^{\frac{1}{m}}$  times, and we continue this procedure until all jobs are assigned to a processor.

---

**Algorithm 3.**


---

- 1:  $i = 1; \mathcal{J}_i = \mathcal{J};$
  - 2: **repeat**
  - 3:   Run the algorithm in [16] for the problem  $1|r_j, d_j, pmtn|E$  with input the set of jobs in  $\mathcal{J}_i$  and get the preemptive schedule  $\mathcal{S}_{pr,i}$ ;
  - 4:   Create the tree-representation graph  $T_i$  of  $\mathcal{S}_{pr,i}$ ;
  - 5:   Let  $\mathcal{J}_{i+1}$  be the set of jobs (vertices) with at least  $n^{\frac{1}{m}}$  children in  $T_i$ ;
  - 6:   For each job  $J_j \in \mathcal{J}_i \setminus \mathcal{J}_{i+1}$ , schedule  $J_j$  on the processor  $i$  in its largest interval in  $\mathcal{S}_{pr,i}$  and get the non-preemptive schedule  $\mathcal{S}_{npr,i}$  for the processor  $i$ ;  $i = i + 1$ ;
  - 7: **until**  $\mathcal{J}_i \neq \emptyset$
  - 8: **return**  $\mathcal{S}_{npr}$  which is the union of  $\mathcal{S}_{npr,i}$ 's;
- 

**Theorem 3.** *Algorithm 3 achieves an approximation ratio of  $m^\alpha(\sqrt[m]{n})^{\alpha-1}$  for  $P|r_j, d_j|E$ .*

*Proof.* Let  $n_i = |\mathcal{J}_i|$  be the number of jobs in the  $i$ -th iteration. We will first show that in iteration  $i$  there are at most  $n_i^{1-\frac{1}{m}}$  vertices with at least  $n^{\frac{1}{m}}$  children. Assume that there were at least  $n_i^{1-\frac{1}{m}}$  vertices with at least  $n^{\frac{1}{m}}$  children. Then the number of children in the graph  $T_i$  is at least  $n_i^{1-\frac{1}{m}} \cdot n^{\frac{1}{m}} \geq n_i$ , which is a contradiction. Let  $k$  be the number of the iterations of the algorithm. By the previous observation, we have that  $k \leq m$ .

Consider the  $i$ -th iteration. Each job  $J_j \in \mathcal{J}_i \setminus \mathcal{J}_{i+1}$  has strictly less than  $n^{\frac{1}{m}}$  children in  $T_i$ , and hence by Lemma 2 it is preempted strictly less than  $n^{\frac{1}{m}}$  times in  $\mathcal{S}_{pr,i}$ . Our algorithm schedules  $J_j$  in  $\mathcal{S}_{npr,i}$  during its largest interval in  $\mathcal{S}_{pr,i}$ . Thus the speed of  $J_j$  in  $\mathcal{S}_{npr,i}$  is at most  $n^{\frac{1}{m}}$  times the speed of  $J_j$  in  $\mathcal{S}_{pr,i}$ . Therefore, using Proposition 2, for any job  $J_j \in \mathcal{J}_i \setminus \mathcal{J}_{i+1}$  it holds that  $E(\mathcal{S}_{npr,i}, J_j) \leq (\sqrt[m]{n})^{\alpha-1} \cdot E(\mathcal{S}_{pr,i}, J_j)$ . For the energy consumed by  $\mathcal{S}_{npr}$  we have

$$E(\mathcal{S}_{npr}) = \sum_{i=1}^k \sum_{J_j \in \mathcal{J}_i \setminus \mathcal{J}_{i+1}} E(\mathcal{S}_{npr,i}, J_j) \leq (\sqrt[m]{n})^{\alpha-1} \cdot \sum_{i=1}^k \sum_{J_j \in \mathcal{J}_i \setminus \mathcal{J}_{i+1}} E(\mathcal{S}_{pr,i}, J_j)$$

Note that the schedule  $\mathcal{S}_{pr,i}$  is the optimal preemptive schedule for the jobs in  $\mathcal{J}_i$ , while the schedule  $\mathcal{S}_{pr,1}$  is the optimal preemptive schedule for the jobs in  $\mathcal{J}_1$ . As  $\mathcal{J}_i \subset \mathcal{J}_1 = \mathcal{J}$  it holds that

$$\sum_{J_j \in \mathcal{J}_i \setminus \mathcal{J}_{i+1}} E(\mathcal{S}_{pr,i}, J_j) \leq \sum_{J_j \in \mathcal{J}_i} E(\mathcal{S}_{pr,i}, J_j) \leq \sum_{J_j \in \mathcal{J}} E(\mathcal{S}_{pr,1}, J_j)$$

Therefore, we get that

$$E(\mathcal{S}_{npr}) \leq (\sqrt[m]{n})^{\alpha-1} \cdot \sum_{i=1}^k \sum_{J_j \in \mathcal{J}} E(\mathcal{S}_{pr,1}, J_j) \leq m \cdot (\sqrt[m]{n})^{\alpha-1} \cdot \sum_{J_j \in \mathcal{J}} E(\mathcal{S}_{pr,1}, J_j)$$

Note that  $\sum_{J_j \in \mathcal{J}} E(\mathcal{S}_{pr,1}, J_j)$  is the optimal energy consumption if all jobs are executed preemptively on a single processor. By Proposition 3 and since the energy consumption of an optimal multiprocessor preemptive schedule without migrations is a lower bound to the energy consumption of an optimal multiprocessor non-preemptive schedule (without migrations)  $\mathcal{S}^*$ , we have that

$$E(\mathcal{S}_{npr}) \leq m^\alpha \cdot (\sqrt[m]{n})^{\alpha-1} \cdot E(\mathcal{S}^*)$$

and the theorem follows.  $\square$

## References

1. Albers, S.: Energy-efficient algorithms. *Communications of ACM* 53, 86–96 (2010)
2. Albers, S., Antoniadis, A., Greiner, G.: On multi-processor speed scaling with migration: extended abstract. In: SPAA, pp. 279–288. ACM (2011)
3. Albers, S., Müller, F., Schmelzer, S.: Speed scaling on parallel processors. In: SPAA, pp. 289–298. ACM (2007)
4. Angel, E., Bampis, E., Kacem, F., Letsios, D.: Speed scaling on parallel processors with migration. In: Kaklamanis, C., Papatheodorou, T., Spirakis, P.G. (eds.) Euro-Par 2012. LNCS, vol. 7484, pp. 128–140. Springer, Heidelberg (2012)
5. Antoniadis, A., Huang, C.-C.: Non-preemptive speed scaling. In: Fomin, F.V., Kaski, P. (eds.) SWAT 2012. LNCS, vol. 7357, pp. 249–260. Springer, Heidelberg (2012)
6. Bampis, E., Letsios, D., Lucarelli, G.: Green scheduling, flows and matchings. In: Chao, K.-M., Hsu, T.-S., Lee, D.-T. (eds.) ISAAC 2012. LNCS, vol. 7676, pp. 106–115. Springer, Heidelberg (2012)
7. Bampis, E., Letsios, D., Milis, I., Zois, G.: Speed scaling for maximum lateness. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 25–36. Springer, Heidelberg (2012)
8. Bingham, B.D., Greenstreet, M.R.: Energy optimal scheduling on multiprocessors with migration. In: ISPA, pp. 153–161. IEEE (2008)
9. Bunde, D.P.: Power-aware scheduling for makespan and flow. In: SPAA, pp. 190–196. ACM (2006)
10. Chan, H.-L., Chan, W.-T., Lam, T.W., Lee, L.-K., Mak, K.-S., Wong, P.W.H.: Energy efficient online deadline scheduling. In: SODA, pp. 795–804 (2007)
11. Chen, J.-J., Hsu, H.-R., Chuang, K.-H., Yang, C.-L., Pang, A.-C., Kuo, T.-W.: Multiprocessor energy-efficient scheduling with task migration considerations. In: ECTRS, pp. 101–108 (2004)
12. Greiner, G., Nonner, T., Souza, A.: The bell is ringing in speed-scaled multiprocessor scheduling. In: SPAA, pp. 11–18. ACM (2009)
13. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM* 34, 144–162 (1987)

14. Li, M., Yao, F.F.: An efficient algorithm for computing optimal discrete voltage schedules. *SIAM Journal on Computing* 35, 658–671 (2006)
15. Pruhs, K., van Stee, R., Uthaisombut, P.: Speed scaling of tasks with precedence constraints. *Theory of Computing Systems* 43, 67–80 (2008)
16. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: FOCS, pp. 374–382 (1995)

# Selection from Read-Only Memory with Limited Workspace

Amr Elmasry<sup>1</sup>, Daniel Dahl Juhl<sup>2</sup>, Jyrki Katajainen<sup>2</sup>,  
and Srinivasa Rao Satti<sup>3,\*</sup>

<sup>1</sup> Department of Computer Engineering and Systems, Alexandria University  
Alexandria 21544, Egypt

<sup>2</sup> Department of Computer Science, University of Copenhagen  
Universitetsparken 5, 2100 Copenhagen East, Denmark

<sup>3</sup> School of Computer Science and Engineering, Seoul National University  
599 Gwanakro, Gwanak-Gu, Seoul 151-744, Korea

**Abstract.** In the classic selection problem the task is to find the  $k$ th smallest of  $N$  elements. We study the complexity of this problem on a space-bounded random-access machine: The input is given in a read-only array and the capacity of workspace is limited. We prove that the linear-time prune-and-search algorithm—presented in most textbooks on algorithms—can be adjusted to use  $O(N)$  bits instead of  $\Theta(N)$  words of extra space. Prior to our work, the best known algorithm by Frederickson could perform the task with  $O(N)$  bits of extra space in  $O(N \log^* N)$  time. In particular, our result separates the space-restricted random-access model and the multi-pass streaming model (since we can bypass the  $\Omega(N \log^* N)$  lower bound known for the latter model). We also generalize our algorithm for the case when the size of the workspace is  $O(S)$  bits,  $\lg^3 N \leq S \leq N$ . The running time of our generalized algorithm is  $O(N \lg^*(N/S) + N \lg N / \lg S)$ , slightly improving over the bound  $O(N \lg^*(N \lg N/S) + N \lg N / \lg S)$  of Frederickson’s algorithm. Of independent interest, the wavelet stack—a structure we used for repeated pruning—may also be useful in other applications.

## 1 Introduction

Let  $x_1, x_2, \dots, x_N$  be the set of input elements. In the selection problem we want to find the  $k$ th smallest of these elements. Without loss of generality, we can assume that the elements are distinct (since in the case of equal elements the indices can be used to distinguish the elements). That is, the output will be a single index  $m$  with a guarantee that  $k - 1$  elements are smaller than  $x_m$  and  $N - k$  elements are larger than  $x_m$ .

In the unrestricted random-access machine the asymptotic complexity of the selection problem was settled to be  $\Theta(N)$  by Blum et al. [3] in their celebrated

---

\* Research partly supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (grant number 2012-0008241).

article from 1973. Here we study the problem in the *space-restricted random-access model*: The input is given in a read-only array and only a limited amount of additional workspace is available. We focus on the case where the amount of available space is  $O(N)$  bits. Surprisingly, although the selection problem with space restriction has been studied in several papers [8,13,15,19], its exact complexity is still not fully resolved—even after our study.

To get a feeling of the difficulties encountered when designing algorithms for this model of computation, let us consider an algorithm that solves the selection problem using  $O(\lg N)$  extra bits (a constant number of machine words). As in many other algorithms, we maintain two indices that specify the so-called *filters*; the elements whose values fall within the range of the two filters are still possible candidates for the  $k$ th smallest element. We say that the elements within the range of the filters are *alive*. Before proceeding, the input is scanned once to make the minimum and the maximum elements the initial filters.

When the procedure is called, the elements are in a contiguous segment of memory and only those elements that are alive will be considered. Assume that the size of the segment is  $M$ . First, we divide the segment into two zones: the first  $\lfloor M/2 \rfloor$  elements form the first zone and the remaining elements the second zone. Second, we check which of the zones contains the majority of alive elements (this idea is from [15]); we say that this zone is *heavy*. Third, we select the median of the heavy zone recursively. Let  $x_\ell$  and  $x_r$  be the filters and  $x_m$  the median found. After the recursive call, we scan through the elements in the current segment to determine whether the  $k$ th smallest element is in the interval  $(x_\ell \dots x_m)$ , is equal to  $x_m$ , or is in the interval  $(x_m \dots x_r)$ . If  $x_m$  is the  $k$ th smallest element, we return  $x_m$  as output. In the two other cases, we update the filters and set  $k$  to  $k - j$  if  $j$  smaller alive elements were eliminated. Because of the median finding in the heavy zone, at least one fourth of the alive elements will be removed from further consideration. Finally, we recursively find the  $k$ th smallest of the remaining alive elements in the whole segment.

We keep information about which of the two subproblems is called and which of the two zones is heavy in a recursion stack. The boundaries of the segment of the caller can be computed from the boundaries of the callee using these additional bits. The filters are passed from the caller to the callee, and vice versa. That is, the workspace is  $O(\lg N)$  bits.

In the analysis of the running time, we use  $A$  to denote the number of alive elements and  $M$  the size of the contiguous segment where these elements reside. Now the worst-case running time can be described using the recurrence:

$$T(A, M) \leq \begin{cases} c_1 \cdot M & \text{if } A < 10 \\ T(A', \lceil M/2 \rceil) + T(A'', M) + c_2 \cdot M & \text{if } A \geq 10, \end{cases}$$

where  $c_1$  and  $c_2$  are positive constants, and  $A'$  and  $A''$  denote the number of alive elements of the subproblems in the first and second recursive calls, respectively. We know that  $\lceil A/2 \rceil \leq A' \leq \lceil M/2 \rceil$  and  $A'' \leq \lfloor 3A/4 \rfloor$ . We encourage the reader to solve this recurrence. (An answer can be found on the last page of this paper.)

This algorithm highlights several aspects that are important for algorithms designed for the space-bounded random-access machine. Since we cannot move

**Table 1.** The best known algorithms for selecting the  $k$ th smallest of  $N$  elements in the space-restricted random-access model;  $N$  is the size of the read-only input. The first three algorithms work for a larger possible range of workspace, but we give their running times only for these specific values.

Inventors	Workspace in bits	Running time
Munro and Raman [15]	$\Theta(\lg N)$	$O(N^{1+\varepsilon})$
Raman and Ramnath [19]	$\Theta(\lg^2 N)$	$O(N \lg^2 N)$
Frederickson [8]	$\Theta(\lg^3 N)$	$O(N \lg N / \lg \lg N)$
Frederickson [8]	$\Theta(N)$	$O(N \log^* N)$
Blum et al. [3]	$\Theta(N \lg N)$	$\Theta(N)$
This paper	$O(N)$	$\Theta(N)$

and modify elements, we have to scan the read-only array several times. The elements that are alive are scattered over a large area, so we have to scan over already eliminated elements several times. Due to the limited memory resources, we cannot store information and we have to recompute some information that has been computed before. Also, because of the limited workspace we have to resort to some bit tricks to save space.

Asymptotically, the algorithm described above is not the fastest known when the amount of workspace is  $O(\lg N)$  bits. The performance of the best known selection algorithms is summarized in Table 1. In the current paper we improve the known results when the amount of extra space is  $O(N)$  bits by giving a new implementation for the algorithm of Blum et al. [3]. For the general case of  $O(S)$  bits of workspace, the best known algorithm is that of Frederickson [8]. The running time of Frederickson's algorithm is  $O(N \lg^*(N \lg N/S) + N \lg N / \lg S)$  when  $S = \Omega(\lg^3 N)$ . We generalize our algorithm to use  $O(S)$  bits of workspace, and improve the running time to  $O(N \lg^*(N/S) + N \lg N / \lg S)$ .

In the literature two different models of computation have been considered: the multi-pass streaming model [4,8,13] and the space-restricted random-access model (that is used in this paper) [8,15,19]. The essential difference is that in the streaming model the read-only input must be accessed sequentially, but multiple scans of the entire input are allowed. In addition to the running time, the number of scans performed would be another optimization target. Chan [4] proved that Frederickson's algorithm is asymptotically optimal for the selection problem in the multi-pass streaming model. He questioned whether this lower bound would also hold in the space-restricted random-access model. We answer this question by bypassing this bound on the space-bounded random-access machine.

As should be obvious, we rely heavily on the random-access capabilities. The kernel of our construction is the wavelet stack—a new data structure that allows us to eliminate elements while being able to sequentially scan the alive elements and jump over the eliminated ones. In the meantime, such structure only requires a constant number of bits per element (instead of the usual  $\lceil \lg N \rceil$  bits required for storing indices). The wavelet stack is by no means restricted to this particular application; our hope is that it would be generally useful for prune-and-search algorithms in the space-bounded setting. A wavelet stack comprises several layers

of bit vectors, each supporting *rank* and *select* queries in  $O(1)$  worst-case time [6,11,14,18]. Using the *rank* and *select* facilities, we can navigate between the layers of the stack and perform successor queries efficiently.

## 2 Basic Tools: Bit Vectors with *rank* and *select* Support

In this section we briefly describe the set of basic tools used by us. The reader is advised to check the original references if our descriptions are too short and more details are needed.

A *bit vector* is an array of bits (0's and 1's). In our case a bit vector  $V$  is a data structure that has a fixed size and supports the operations:

$V.access(i)$ : Return the bit at index  $i$ , also denoted as  $V[i]$ .

$V.rank(i)$ : Return the number of 1-bits among the bits  $V[0], V[1], \dots, V[i]$ .

$V.select(j)$ : Return the index of the  $j$ th 1-bit, i.e. when the return value is  $i$ ,  $V[i] = 1$  and  $rank(i) = j$ .

Let  $w$  denote the size of the machine word in bits. It is a routine matter [12, Section 7.1.3] to store a bit vector of size  $N$  such that it occupies  $\lceil N/w \rceil$  words and any consecutive substring of at most  $w$  bits—not only a single bit—can be accessed in  $O(1)$  worst-case time.

There exist several space-efficient solutions to support the two other operations in  $O(1)$  worst-case time. Jacobson [11] showed how to support *rank* and *select* in  $O(\lg N)$  bit probes using only  $o(N)$  bits in addition to the bit vector itself. Clark and Munro [6,14] showed how to support the queries in  $O(1)$  worst-case time on a random-access machine with word size  $\Theta(\lg N)$ ; Raman et al. [18] improved the space bound to  $O(N \lg \lg N / \lg N)$  bits, which was shown by Golynski [9] to be optimal provided that the bit vector is stored in plain form. The basic idea in all the mentioned solutions is to divide the input into blocks, store the *rank* and *select* values for some specific positions, and compute the *rank* and *select* values for the remaining positions on the fly using the stored values, values in some precomputed tables, and bits in the original bit vector.

Note that the only requirements on the bit vectors we use are that operations must have  $O(1)$  worst-case cost, a space usage must be  $O(N)$  bits, and the construction of the supporting structures must take linear worst-case time. For these requirements, Chazelle [5] described a simple solution to support *rank* operations. After breaking the bit vector into words, for the first bit of each word a *landmark* is computed which is the number of 1-bits preceding this position. Let the words be  $B_0, B_1, \dots, B_{\lceil N/w \rceil - 1}$  and the landmarks  $L_0, L_1, \dots, L_{\lceil N/w \rceil - 1}$ . To compute  $rank(i)$ , we locate the corresponding word  $B_j$ , i.e.  $j = \lfloor i/w \rfloor$ , and the offset  $f$  inside this word, i.e.  $f = i - w \times \lfloor i/w \rfloor$ . Then we mask the bits up to index  $f$  in  $B_j$  and calculate the number of 1-bits in the masked part; let this number be  $r$ . As the end result, we output  $L_j + r$  as  $rank(i)$ . The only difficult part is to calculate the number of 1-bits in a word, but fortunately this can be done by using the population-count function that is a hardware primitive in most modern processors. As far as we know, no equally simple data structure is known to support *select* operations.

### 3 Expanding the Toolkit: Wavelet Stacks

The computation of a recursive algorithm can be described as a tree. A path from the root of the tree (corresponding to the original problem) to the present node (corresponding to the subproblem being currently solved) summarizes the decisions made by the algorithm when exploring the tree. In a prune-and-search algorithm, where we repeatedly eliminate some of the answer candidates, the set of alive elements can be compactly represented using a bit vector. The computational history of the decisions made by such an algorithm can be conveniently described using a stack of bit vectors. We call this kind of data structure a wavelet stack because of its resemblance to wavelet trees [10] (for a survey, see [16]). In this section we describe the data structure in details, and in the next section we show how it can be used to solve the selection problem. We expect this data structure to be useful in other applications as well.

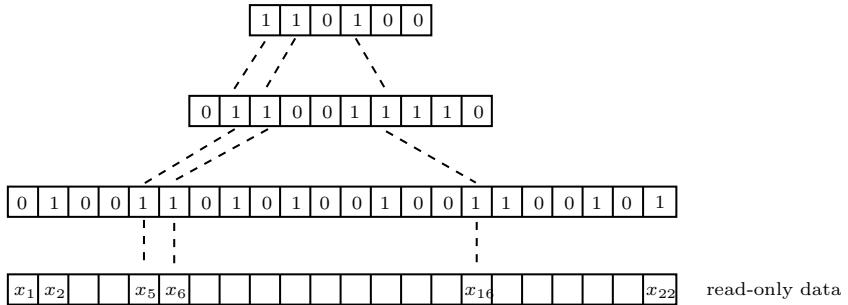
Let  $x_1, x_2, \dots, x_N$  be a sequence of  $N$  elements given in a read-only array. Assume that we want to find some specific subset of these elements using prune-and-search elimination. A prune-and-search algorithm is a recursive procedure that may call itself several times. Hence, we need a recursion stack to keep track of the subproblems being solved. In addition to a recursion stack (with constant-size activation records), we maintain a stack of bit vectors to mark the elements in the current configuration; a 1-bit indicates that the corresponding element is still alive. Subsequently, we can avoid scanning the pruned elements.

In an abstract form, our stack  $H$  of bit vectors—called a *wavelet stack*—is a data structure that can efficiently answer two types of queries:

- $H.\text{alive?}(i)$ : Return whether the element  $x_i$  is alive at the current configuration.
- $H.\text{index}(j)$ : Return the index of the  $j$ th alive element, i.e. the index of the element corresponding to the  $j$ th 1-bit at the top-most bit vector.

To fully understand these operations, we have to consider a concrete implementation of a wavelet stack (see Fig. 1). A wavelet stack is a hierarchy of bit vectors. The bottom-most level stores one bit per element, so at the beginning all elements are potential answers. If we have  $z$  1-bits at level  $h$ , the bit vector at level  $h + 1$  is of size  $z$ . Therefore, the bit vectors become smaller and smaller as we eliminate more elements from further consideration.

The two operations have a nice symmetry: *alive?* traverses up from the bottom to the top of the stack, and *index* traverses down from the top to the bottom of the stack. To implement *alive?*( $i$ ), we start from the bottom-most level and compute *rank*( $i$ ). This gives us the index to access the bit vector above. Continuing upwards and relying on *rank*, we either reach a level where the bit corresponding to the index value is 0 indicating that the element  $x_i$  is not alive any more, or we reach the top-most level where the bit value is 1 indicating that  $x_i$  is still alive. To implement *index*( $j$ ), we start from the top-most level and compute *select*( $j$ ). Then we use the returned index at the level below. This way, we can proceed down using *select* until we reach the bottom-most level. Using the last returned index, we can access the desired element whenever needed.



**Fig. 1.** A wavelet stack for an array of 22 elements. Only elements  $x_5$ ,  $x_6$ , and  $x_{16}$  are alive at this point.

We can summarize the space requirements of the data structure and the time performance of the operations as follows:

**Theorem 1.** *Assume that we have built a wavelet stack of height  $h$  for a read-only array of  $N$  elements. Furthermore, assume that at each level we have succeeded in eliminating a constant fraction of the elements.*

1. *The data structure requires  $O(N)$  bits in total.*
2. *The total time used in the construction of the data structure is  $\Theta(N)$ .*
3. *In the worst case, both `alive?` and `index` operations take  $O(h)$  time.*

*Proof.* Since the number of bits needed at each level is only a constant fraction of that needed at the level below, for a constant  $c < 1$ ,  $\sum_{i=0}^{h-1} c^i N = O(N)$  is an upper bound on the number of bits used. Since the length of the bit vectors is not known beforehand and since their sizes may vary, we can allocate a header storing references to a big bit vector that contains the bits stored at all levels together. This header will only require  $O(\lg^2 N)$  bits.

The construction of a bit vector, including the supporting structures, can be done in linear time. The construction time of the wavelet stack can also be expressed as a geometric series, and is thus  $\Theta(N)$ . Since the structure has  $h$  levels, and the `rank` and `select` operations take  $O(1)$  worst-case time at each level, the  $O(h)$  time bound for `alive?` and `index` follows.  $\square$

## 4 Selection with $O(N)$ Bits

In this section we show how to adapt the prune-and-search algorithm of Blum et al. [3] (alternatively see [7, Section 9.3]) such that it only requires  $O(N)$  bits of space—instead of  $\Theta(N)$  words—but still runs in  $\Theta(N)$  worst-case time.

The basic idea in the algorithm is to search for an element from the set of possible answers and use it to make the set of candidates smaller. This is done repeatedly until the final answer is found. In the variant considered here we use a wavelet stack to keep track of the decisions made by the algorithm. The  $k$ th smallest among  $M$  alive elements is found as follows.

1. A new bit vector  $V$  is pushed onto the top of the wavelet stack; the size of the bit vector equals the number of the currently alive elements  $M$ .
2. Divide the set of  $M$  elements into groups of five, so that only the last group may have less than 5 elements. Find the median of each of the  $\lceil M/5 \rceil$  groups. After processing a group, mark its median as alive and the other elements of the group as not alive in the top-most bit vector  $V$ .
3. Recursively compute the median  $x$  of these medians.
4. Set the bits of  $V$  to indicate that all the  $M$  elements are again alive.
5. Scan through the alive elements and determine how many of them are smaller than  $x$  and how many are larger. Let these numbers be  $\sigma$  and  $\tau$ , respectively. If  $k = \sigma + 1$ , i.e. if  $x$  is the  $k$ th smallest element, stop and return  $x$  as an answer. If  $k \leq \sigma$ , mark the the elements smaller than  $x$  as the only alive elements in  $V$  and recursively compute the  $k$ th smallest of these elements. Otherwise, if  $k > \sigma + 1$ , mark the elements larger than  $x$  as the only alive elements in  $V$  and set  $k$  to  $k - \sigma - 1$  before the recursive call.
6. After the last recursive call, before returning the answer further to the caller, pop the top-most bit vector  $V$  of the wavelet stack.

This algorithm is a perfect example of recursion in action. There are two recursive calls, one at Step 3 and another at Step 5. The analysis of this algorithm is almost identical to that of the original. The key point is that, even though the input is in a read-only array, we do not waste time in browsing the elements that have already been eliminated as we rely on the *rank-select* facilities provided for the bit vectors. The only overhead is when we want to access an element, we have to traverse down the wavelet stack.

Suppose that we resolve subproblems smaller than 100 without recursive calls, e.g. by having a constant-size array of indices to the elements and applying mergesort indirectly via this array. Let  $T(h, M)$  denote the worst-case running time of the algorithm for a subproblem having size  $M$  when the height of the wavelet stack is  $h$ . An upper bound on the worst-case running time can be expressed using a recurrence relation:

$$T(h, M) \leq \begin{cases} c_1 \cdot h & \text{if } M < 100 \\ c_2 \cdot h \cdot M + T(h+1, M') + T(h+1, M'') & \text{if } M \geq 100, \end{cases}$$

where  $c_1$  and  $c_2$  are some constants, and  $M'$  and  $M''$  denote the sizes of the subproblems in the first and second recursive calls, respectively. We know that  $M' = \lceil M/5 \rceil$  and that  $M'' \leq 7M/10 + 6$  [7, Section 9.3]. For every  $M \geq 100$ ,  $7M/10 + 6 \leq 22M/30$  and  $\lceil M/5 \rceil \leq 7M/30$ . Using these facts, we can solve the recurrence by considering the corresponding recursion tree. The sum of the sizes of the subproblems at the  $i$ th level of the recursion tree,  $i \geq 1$ , is  $(29/30)^{i-1}N$ . In accordance, the cost accompanying the  $i$ th level is  $O(i \cdot (29/30)^{i-1}N)$ . Summing the costs for all the levels, it follows that  $T(1, N) = O(N)$ . The intuition behind the result is that, in spite of the overhead caused by the traversals in the wavelet stack, which increases as a linear function of  $h$ , the size of the subproblems decreases exponentially with  $h$ .

Since the size of the subproblems is the same as for the original algorithm, the total size of all bit vectors constructed is  $O(N)$  too. In addition to the wavelet stack, we need a recursion stack to keep track of the type of the subproblems being solved. However, the depth of recursion is only logarithmic so the recursion stack never uses more than  $O(\lg^2 N)$  bits.

The performance of the algorithm is summarized in the following theorem:

**Theorem 2.** *The  $k$ th smallest of  $N$  elements in a read-only array can be found in  $\Theta(N)$  time using  $O(N)$  extra bits in the worst case.*

## 5 General Solution with $O(S)$ Bits

In this section we extend our algorithm to handle the more general case of using a workspace of  $O(S)$  bits, where  $\lg^3 N \leq S \leq N$ . The main idea is to use Frederickson's algorithm [8] to prune the elements and stop its execution when the number of alive elements is at most  $S$ . To finish the selection process, we resume pruning using an  $O(N)$ -time algorithm that we shall present next.

First we mention the following lemma about Frederickson's algorithm, and omit its proof from this version of the paper. We also refer the reader to [8].

**Lemma 1.** *By applying a trimmed execution of Frederickson's algorithm, we can prune the elements until the number of alive elements is at most  $S$  in  $O(N \lg^*(N/S))$  worst-case time, assuming  $S = \Omega(\lg^3 N)$ .*

If  $S \leq \sqrt{N \lg N}$ , we simply use Frederickson's algorithm all the way. The resulting running time is  $O(N \lg^*((N \lg N)/S) + N \lg N / \lg S) = O(N \lg^*(N/S) + N \lg N / \lg S)$  as claimed. From now on we assume that  $S > \sqrt{N \lg N}$ . We apply a trimmed execution of Frederickson's algorithm as specified by Lemma 1. The outcome is two filters that guard the—at most— $S$  candidates. Consequently, we are left with the task of selecting the designated element among those candidates.

Using a wavelet stack and a bit vector supporting *rank* and *select* queries, we can finish the pruning in  $O(N)$  time. We create and maintain a wavelet stack  $H$ —an element hierarchy where each bit corresponds to an element among those whose values fall between the filters. Since there are at most  $S$  such elements, the wavelet stack  $H$  uses  $O(S)$  bits. While our algorithm is in action, the wavelet stack is to be updated to indicate the elements that are currently surviving the pruning phases. We divide the input sequence (consisting of  $N$  elements) into  $S$  buckets, where the  $u$ th bucket consists of the elements from the input sequence with indices from the range  $[u \cdot \lceil N/S \rceil \dots (u+1) \cdot \lceil N/S \rceil - 1]$ , for  $0 \leq u \leq S-1$  (except possibly the last bucket). In addition, we create the *count vector*  $C$ —a static bit vector that indicates the number of candidates originally contained in each bucket after the execution of Frederickson's algorithm. The count vector  $C$  should efficiently support *rank* and *select* queries. We store these counts encoded in unary, using a 0-bit to mark the border between every two consecutive buckets. Since a total of at most  $S$  candidates need to be stored, the count vector  $C$  contains at most  $S$  ones; and since we have exactly  $S$  buckets,  $C$  contains  $S-1$  zeros. The count vector thus uses  $O(S)$  bits as well.

We can now iterate efficiently through the alive candidates. Let  $i - 1$  be the rank of the element that has just been considered in our iterative scan within the alive elements. First, we find the index  $j$  of the next element to be considered within the wavelet stack. For that we set

$$j = H.\text{index}(i),$$

which is the index of the element we are looking for with respect to those falling between the two filters inherited from Fredrickson's algorithm. Since the difference between the index of a bit, in the count vector  $C$ , and its rank is exactly the index of the bucket its corresponding element belongs to, we can compute the index  $u$  (first bucket has index 0) of the bucket containing this element as

$$u = C.\text{select}(j) - j.$$

We then calculate the index  $t$  that corresponds to the 0-bit resembling the border between the  $u$ th and  $(u - 1)$ th buckets. This is done by setting  $t = \bar{C}.\text{select}(u)$ , where  $\bar{C}$  is the complement vector of  $C$ . Since the total number of elements, among those surviving Frederickson's algorithm, stored in the  $u$  preceding buckets to the current one can be computed by a *rank* query for  $t$  within  $C$ , we determine the position  $p$  of the sought element among those candidates within the surviving elements of the  $u$ th bucket as:

$$p = j - C.\text{rank}(t).$$

If the element that has just been reported was also from bucket  $u$ , we continue scanning the elements of the  $u$ th bucket from where we stopped. Otherwise, we jump to the beginning of the  $u$ th bucket, i.e. to the element whose index is  $u \cdot \lceil N/S \rceil$  in the input array. We sequentially scan the elements of the located bucket, discard the ones falling outside the filters, and count the others until locating the  $p$ th element among them; this is the element we are looking for.

We can now proceed as in the  $O(N)$ -bit solution. Starting with the elements surviving Frederickson's algorithm, we recursively determine the median-of-medians and use it to perform the pruning. During this process we keep the wavelet stack up to date as before. The pruning process continues until only one bucket remains, at such point only  $O(N/S)$  elements are alive. Since this procedure of the algorithm is employed only when  $S = \Omega(\sqrt{N \lg N})$ , the indices of the alive elements can fit in the allowable workspace, each in  $O(\lg N)$  bits, and we continue the selection in linear time.

Since we are operating on buckets, we might have to spend  $\Theta(N/S)$  time for scanning per bucket. However, we note that initially there is at most  $S$  candidates and accordingly at most  $S$  buckets. Since we prune a constant fraction of the candidates in each iteration, we also reduce the bound on the number of the remaining buckets (those having at least one alive element each) by the same constant fraction. Noting that we skip the buckets that have no alive elements, the work done per pass to iterate over the buckets that have at least one alive element can be bounded, as elaborated in the next lemma.

**Lemma 2.** *Given a read-only input array  $X$ , where  $|X| = N$ , and two filters  $f_1$  and  $f_2$ , such that the element to be selected lies in  $I = \{e | f_1 \leq e \leq f_2, e \in X\}$  and  $|I| \leq S$ ; that is, at most  $S$  elements lie between the filters. If  $S = \Omega(\sqrt{N \lg N})$ , we can solve the selection problem in  $O(N)$  time.*

*Proof.* In each pruning iteration we spend time proportional to the number of buckets remaining, while scanning the elements in these buckets and comparing them with the filters. The number of alive elements after we apply the  $i$ th pruning iteration of the median-of-medians algorithm is  $O(S/c^i)$ , for some constant  $c > 1$ . Obviously, the number of buckets that have alive elements cannot exceed the number of elements. It follows that, throughout all the passes of the algorithm, the number of scanned buckets is at most  $O(\sum_{i \geq 0} S/c^i) = O(S)$ . Accordingly, the overall work done in scanning these buckets is  $O(N)$ . Once we have  $O(N/S)$  elements remaining, as  $S = \Omega(\sqrt{N \lg N})$ , we can continue the selection process in the working memory in  $O(N/S)$  time.  $\square$

The main result of this paper is summarized in the upcoming theorem.

**Theorem 3.** *Given a read-only array of  $N$  elements and a workspace of  $O(S)$  bits such that  $\lg^3 N \leq S \leq N$ , it is possible to solve the selection problem in  $O(N \lg^*(N/S) + N \lg N / \lg S)$  worst-case time in the space-restricted random-access model.*

Theorem 3 implies that, in the read-only space-limited setting, Chan’s lower bound [4] for the selection problem in the multi-pass streaming model does not apply to the space-restricted random-access model. This, in turn, indicates that the space-restricted random-access model is more powerful than the multi-pass streaming model.

## 6 Conclusions

We showed that, given an array of  $N$  elements in a read-only memory, the  $k$ th smallest element can be found in  $\Theta(N)$  worst-case time using  $O(N)$  bits of extra space. We also generalized our algorithm to run in  $O(N \lg^*(N/S) + N \lg N / \lg S)$  time using a workspace of  $O(S)$  bits,  $\lg^3 N \leq S \leq N$ . Our main purpose was to show that the lower bound proved by Chan [4] for the multi-pass streaming model can be bypassed in the space-restricted random-access model.

In the read-only setting the selection problem has been studied since 1980 [13]. In contrast to sorting, the exact complexity of selection is still open. The space-time trade-off for sorting is known to be  $\Theta(N^2/S + N \lg S)$  [2,17], where  $S$  is the asymptotic target for the size of the workspace in bits,  $\lg N \leq S \leq N/\lg N$ . The optimal bound for sorting can even be realized using a natural priority-queue-based algorithm [1].

## References

- Asano, T., Elmasry, A., Katajainen, J.: Priority queues and sorting for read-only data. In: Chan, T.-H.H., Lau, L.C., Trevisan, L. (eds.) TAMC 2013. LNCS, vol. 7876, pp. 32–41. Springer, Heidelberg (2013)

2. Beame, P.: A general sequential time-space tradeoff for finding unique elements. *SIAM J. Comput.* 20(2), 270–277 (1991)
3. Blum, M., Floyd, R.W., Pratt, V., Rivest, R.L., Tarjan, R.E.: Time bounds for selection. *J. Comput. System Sci.* 7(4), 448–461 (1973)
4. Chan, T.M.: Comparison-based time-space lower bounds for selection. *ACM Trans. Algorithms* 6(2), 26:1–26:16 (2010)
5. Chazelle, B.: A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.* 17(3), 427–462 (1988)
6. Clark, D.: Compact Pat Trees. Ph.D. thesis, Department of Computer Science, University of Waterloo, Waterloo (1996)
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: *Introduction to Algorithms*, 3rd edn. The MIT Press, Cambridge (2009)
8. Frederickson, G.N.: Upper bounds for time-space trade-offs in sorting and selection. *J. Comput. System Sci.* 34(1), 19–26 (1987)
9. Golyński, A.: Optimal lower bounds for rank and select indexes. *Theoret. Comput. Sci.* 387(3), 348–359 (2007)
10. Grossi, R., Gupta, A., Vitter, J.S.: High-order entropy-compressed text indexes. In: *SODA 2003*, pp. 841–850. SIAM, Philadelphia (2003)
11. Jacobson, G.: Space-efficient static trees and graphs. In: *FOCS 1989*, pp. 549–554. IEEE Computer Society, Los Alamitos (1989)
12. Knuth, D.E.: *Combinatorial Algorithms: Part 1, The Art of Computer Programming*, vol. 4A. Addison-Wesley, Boston (2011)
13. Munro, J.I., Paterson, M.S.: Selection and sorting with limited storage. *Theoret. Comput. Sci.* 12(3), 315–323 (1980)
14. Munro, J.I.: Tables. In: Chandru, V., Vinay, V. (eds.) *FSTTCS 1996*. LNCS, vol. 1180, pp. 37–42. Springer, Heidelberg (1996)
15. Munro, J.I., Raman, V.: Selection from read-only memory and sorting with minimum data movement. *Theoret. Comput. Sci.* 165(2), 311–323 (1996)
16. Navarro, G.: Wavelet trees for all. In: Kärkkäinen, J., Stoye, J. (eds.) *CPM 2012*. LNCS, vol. 7354, pp. 2–26. Springer, Heidelberg (2012)
17. Pagter, J., Rauhe, T.: Optimal time-space trade-offs for sorting. In: *FOCS 1998*, pp. 264–268. IEEE Computer Society, Los Alamitos (1998)
18. Raman, R., Raman, V., Satti, S.R.: Succinct indexable dictionaries with applications to encoding  $k$ -ary trees, prefix sums and multisets. *ACM Trans. Algorithms* 3(4), Article 43 (2007)
19. Raman, V., Ramnath, S.: Improved upper bounds for time-space trade-offs for selection. *Nordic J. Comput.* 6(2), 162–180 (1999)

Answer: By substitution one can show that  $\mathcal{O}(N^2) > T(N - 2, N) \mathcal{O}(N_\varepsilon)$ .

# Determinization of Büchi Automata as Partitioned Automata<sup>★</sup>

Cong Tian<sup>1</sup>, Zhenhua Duan<sup>1,★★</sup>, and Mengfei Yang<sup>2</sup>

<sup>1</sup> ICTT and ISN Lab, Xidian University, Xi'an, 710071, P.R. China

<sup>2</sup> China Academy of Space Technology, Beijing, 100094, P.R. China

**Abstract.** In this paper, Nondeterministic Büchi Automata (NBA) are equivalently transformed as a new kind of deterministic omega automata, Deterministic Partitioned Automata (DPA). Different from the existing automata, at most three different transitions may occur between two states of a DPA. This leads to a determinization construction of NBA with smaller state complexity but larger transition complexity. Compared with the existing determinization constructions of NBA, the new one is intuitive and easily implementable since it is completely based on subset construction. In addition, we have proved that both nondeterministic and deterministic partitioned automata share the same expressive power with NBA.

**Keywords:** Büchi automata, Determinization, Verification, Partitioned automata, Subset construction.

## 1 Introduction

The theory of automata over infinite words underpins much of formal verification. In automata-based model checking [1,2,5,6], to decide whether a given system described by an automaton satisfies a desired property specified by a Büchi automaton [3], one constructs the intersection of the system automaton with the complementation of the property automaton, and checks its emptiness [1,2]. To complement Nondeterministic Büchi automata (NBA), Rabin, Muller, Parity as well as Streett automata [4] are often involved, since deterministic Büchi automata are not closed under complementation. Recently, co-Büchi automata have attracted much attention because of its simplicity and its surprising utility [11,12,13].

Complementation of Büchi automata is hard and has been a forty years investigated question since 60s [7,8]. In 1962, Büchi introduced Büchi automaton and gave a complementation construction with the state complexity being  $2^{2^{O(n)}}$  [3]. In [15], an improved complementation of Büchi's construction was described with  $2^{O(n^2)}$  states. Further, in [9], Safra presented a determinization construction, which also enabled a complementation construction with  $n^{2n}$  state complexity. Friedgut, Kupferman and Vardi presented a new construction with complexity being  $(0.97n)^n$  in 2006 [14]. Finally, Schewe closed this problem by a construction with complexity being  $(0.76n)^n$ .

---

\* This research is supported by the NSFC Grant No. 61003078, 61133001, and 61272117, 973 Program Grant No. 2010CB328102 and ISN Lab Grant No. ISN1102001.

\*\* Corresponding author.

[19] that matches the lower bound proved by Yan [18]. All the existing constructions are analyzed by state complexity without considering the transition complexity. Most often, it is sufficient to analyze the size of an automaton by computing the state space. However, in some circumstance, the state space can be exponentially saved by considerable waste of transitions. Thus, in these cases, the transition complexity cannot be ignored.

In general, determinization makes complementing omega automata easier. Therefore, nearly at the same time when researching on complementing Büchi automata was started, determination of Büchi automaton was also taken into account [16]. In 1966, McNaughton proved that a Büchi automaton can equivalently be transformed into a Deterministic Muller Automaton (DMA) [17]. Subsequently, in 1988, Safra [9] proposed a method to transform a nondeterministic Büchi automaton into a deterministic Rabin or Muller automaton. With this approach, given a nondeterministic Büchi automaton with  $n$  states, the equivalent deterministic automaton has  $12^n n^{2n}$  states. Safra's construction is often difficult. In 1995 Muller and Schupp presented a more intuitive alternative [10]. Piterman [20] presented a tighter construction by utilizing *compact Safra trees* which are obtained by using a dynamic naming technique throughout the construction of Safra trees. With compact Safra trees, a nondeterministic Büchi automaton can be transformed into an equivalent deterministic parity automaton with  $2n^n n!$  states and  $2n$  priorities (can be equivalently transformed to a deterministic Rabin automaton with the same complexity and  $n$  priorities). The advantage of Piterman's determinization is to output deterministic Parity automata which is easier to manipulate. By separating the principal acceptance mechanism from the concrete acceptance condition of a Büchi automaton with  $n$  states, Schewe presented the construction of an equivalent deterministic Rabin transition automaton with  $O((1.65n)^n)$  states, which can be simply translated to a standard Rabin automaton with  $O((2.66n)^n)$  states [22]. Based on history trees, Schewe also obtained an  $O((n!)^2)$  transformation from Büchi automata to deterministic parity automata that factually resembles Piterman's construction (Liu and Wang [21] independently present a similar state complexity result to Piterman's determinization). Schewe's construction is mainly based on a new data structure, namely, history tree which is an ordered tree with labels. Compared to Safra's construction, Schewe's construction is simple and intuitive. Subsequently [23], a lower bound for the transformation from Büchi automata to deterministic Rabin transition automata is proved to be  $O((1.64n)^n)$  [23]. Therefore the state complexity for Schewe's construction of Rabin transition automata is optimal. For the ordinary Rabin acceptance condition, the construction via history trees also conducts the best upper-bound complexity result  $O((2.66n)^n)$ . However, the price paid for that is the use of  $2^{n-1}$  Rabin pairs (instead of  $n$  in Safra's construction). So it is hard to judge the efficiency of the determinization of NBW in Rabin acceptance condition via history trees.

In this paper, a new kind of omega automaton, Partitioned Automaton (PA), is proposed for determinizing a Büchi automaton. Different from the traditional automata, possibly, at most three different transitions occur between two states. This brings to partitioned automaton an essential  $3^n$  times larger transition complexity than the traditional automata. However, by partitioned automaton, a novel construction for determinizing Büchi automaton is obtained with the state complexity being exponentially

smaller than the constructions by the traditional automata such as Rabin, Street, Parity and Muller automata. The new construction is intuitive and easily implementable since it is completely based on subset construction. In addition, the properties of PA are studied, especially, it is proved that Nondeterministic PA (NPA), Deterministic PA (DPA) and NBA have the same expressive power.

The rest parts of the paper are organized as follows. The next section briefly presents preliminaries including the definition of Büchi automata as well as its relevant properties. Section 3 is devoted to introducing the new omega automaton, partitioned automaton. Constructions for determinizing Büchi automaton through partitioned automaton is presented in section 4. While properties of partitioned automaton are studied in section 5. Finally, conclusions are drawn in section 6.

## 2 Preliminaries

Let  $\Sigma$  denote a finite set of symbols called an alphabet. An infinite word  $\alpha \in \Sigma^\omega$  is an infinite sequence of symbols from  $\Sigma$ .  $\Sigma^\omega$  is the set of infinite words over  $\Sigma$ . We present  $\alpha$  as a function  $\alpha : N_0 \rightarrow \Sigma$ , where  $N_0$  is the set of non-negative integers. Thus,  $\alpha(i)$  denotes the letter occurring at the  $i^{\text{th}}$  position. In general,  $\text{Inf}(\alpha)$  denotes the set of symbols from  $\Sigma$  which occur infinitely often in  $\alpha$ . Formally,  $\text{Inf}(\alpha) = \{a \in \Sigma \mid \exists^\omega n \in N_0 : \alpha(n) = a\}$ .

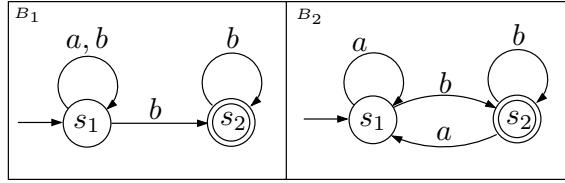
**Definition 1. (Büchi Automaton)** A Büchi automaton over  $\Sigma$  is a tuple  $B = (S, \rightarrow, I, F)$  where  $S$  is a, non-empty, finite set of states,  $I \subseteq S$  is a set of initial states,  $\rightarrow \subseteq S \times \Sigma \times S$  is a transition relation, and  $F \subseteq S$  is the set of accepting states.  $\square$

A run of  $B$  on an infinite words  $\alpha$  is an infinite sequence  $\rho : N_0 \rightarrow S$  such that  $\rho(0) \in I$  and for all  $i \in N_0$ ,  $(\rho(i), \alpha(i), \rho(i+1)) \in \rightarrow$ .  $B$  accepts an input  $\alpha : N_0 \rightarrow \Sigma$  if there is a run  $\rho$  of  $B$  on  $\alpha$  such that  $\text{Inf}(\rho) \cap F \neq \emptyset$ . A Büchi automaton is said to be deterministic if  $I$  is a singleton, and for any  $(s, a, s') \in \rightarrow$ , there exist no  $(s, a, s'') \in \rightarrow$  with  $s''$  being different from  $s'$ .

It is well known that nondeterministic Büchi automata is strictly more powerful than Deterministic Büchi Automata (DBA), i.e. for some NBA  $B_1$  there exists no DBA  $B_2$  such that  $L(B_1) = L(B_2)$ . A widely used example is the  $\omega$ -language  $(a + b)^* \cdot b^\omega$ , which can only be accepted by NBA  $B_1$  in Fig.1, but not by a deterministic one. This induces the failure of the traditional subset construction used to determinize a finite state automaton. When applying subset construction to  $B_1$ , DBA  $B_2$  in Fig.1 is obtained. It is transparent that  $B_2$  will accept  $(a^* \cdot b)^\omega$ , which is a super set of  $(a + b)^* \cdot b^\omega$ . Accordingly, to determinize a Büchi automaton, the traditional subset construction does not work.

**Definition 2. (Muller Automata)** A Muller automaton is over  $\Sigma$  is a tuple  $A = (S, \rightarrow, I, T)$ , where  $S$ ,  $\rightarrow$  and  $I$  are the same as before,  $T = \{T_1, T_2, \dots, T_k\}$  is a set of accepting sets with  $T_i \subseteq S$  for each  $i \in \{1, 2, \dots, k\}$ .  $\square$

An automaton  $A$  accepts an input  $\alpha : N_0 \rightarrow \Sigma$  if there is a run  $\rho$  of  $A$  on  $\alpha$  such that  $\text{Inf}(\rho) \in T$ , that is,  $\text{Inf}(\rho) = T_i$  for some  $i \in \{1, 2, \dots, k\}$ .



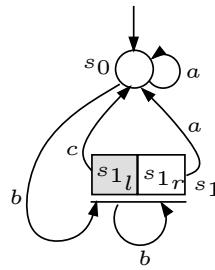
**Fig. 1.** NBA  $B_1$ , and deterministic version of  $B_1$  by subset construction

### 3 Partitioned Automata

Now we present the definition of the new proposed omega automaton, namely Partitioned Automaton (PA). Different from the traditional automata, at most three different transitions may occur between two states in a partitioned automaton.

**Definition 3. (Partitioned Automata)** A partitioned automaton is a tuple  $P = (S, \rightarrow, I, Q)$ , where  $S = S_C \cup S_R \cup S_R^l \cup S_R^r$  is the set of states,  $I \subseteq S_C \cup S_R$  is the set of initial states. Let  $Q' \subseteq S_C$ .  $Q = Q' \cup S_R \cup S_R^l$  denotes the set of accepting states,  $\rightarrow = S \times \Sigma \times (S_C \cup S_R)$  is the set of transitions.  $\square$

In a partitioned automaton, a state in  $S_C$ , called a circle state, is identical to the states in the general automata and denoted by a circle, while a state  $s \in S_R$ , named a rectangle state, is a duple,  $s = (s_l, s_r)$ , and denoted by a rectangle divided in to two sides with an underline. The rectangle is labeled with  $s$ , the left part, called left state of  $s$ , is labeled by  $s_l$  and the right part, called right state of  $s$ , is labeled with  $s_r$ . For convenience, let  $left(s)$  denote the left state  $s_l$  and  $right(s)$  denote the right state  $s_r$  of a rectangle state  $s = (s_l, s_r)$ . Further,  $S_R^l = \{left(s) \mid s \in S_R\}$  and  $S_R^r = \{right(s) \mid s \in S_R\}$  is the set of left states and right states of states in  $S_R$  respectively. Moreover, for a state  $s_l$  in  $S_R^l$ ,  $rec(s_l) = s$ , if  $left(s) = s_l$ , and for  $s_r$  in  $S_R^r$ ,  $rec(s_r) = s$ , if  $right(s) = s_r$ . An example of



**Fig. 2.** A partitioned automaton

partitioned automaton is illustrated in Fig.2, where  $S = \{s_0, s_1, s_{1_l}, s_{1_r}\}$ ,  $s_1 = (s_{1_l}, s_{1_r})$ ,  $I = \{s_0\}$ ,  $\rightarrow = \{(s_1, b, s_1), (s_0, b, s_1), (s_0, a, s_0), (s_{1_l}, c, s_0), (s_{1_r}, a, s_0)\}$  and  $Q = \{s_{1_l}^l, s_1\}$ .

A run of a partitioned automaton on an infinite sequence  $\alpha \in \Sigma^\omega$  is an infinite sequence  $\rho : N_0 \rightarrow S$  such that  $\rho(0) \in I$  and for all  $i \in N_0$ , if  $\rho(i+1) \in S_C \cup S_R$ ,  $(\rho(i), \alpha(i), \rho(i+1)) \in \rightarrow$ , while if  $\rho(i+1) \in S_R^l \cup S_R^r$ ,  $(\rho(i), \alpha(i), rec(\rho(i+1))) \in \rightarrow$ .  $P$

accepts an input  $\alpha : N_0 \rightarrow \Sigma$  if there is a run  $\rho$  of  $P$  on  $\alpha$  such that  $\text{Inf}(\rho) \cap Q \neq \emptyset$ . Accordingly, for the partitioned automaton in Fig.2, sequences  $aaaaaa\dots$  and  $babababababa\dots$  are not acceptable while  $bbbbbb\dots$  and  $bcbcbcfcfc...\dots$  are acceptable.

A partitioned automaton is said to be deterministic if  $I$  is a singleton. Also, for any  $(s, a, s') \in \rightarrow$ , there exists no  $(s, a, s'') \in \rightarrow$  with  $s''$  being different to  $s'$ ; if  $s \in S_R$ , there exists no  $(\text{left}(s), a, s'')$  and  $(\text{right}(s), a, s'') \in \rightarrow$ ; if  $s \in S_R^l$ , there exists no  $(\text{rec}(s), a, s'')$  and  $(\text{right}(\text{rec}(s)), a, s'') \in \rightarrow$ ; and if  $s \in S_R^r$ , there exists no  $(\text{rec}(s), a, s'')$  and  $(\text{left}(\text{rec}(s)), a, s'') \in \rightarrow$ . For instance, the PA in Fig.2 is deterministic. And if an extra transition  $(s_{1r}, b, s_0)$  from state  $s_{1r}$  to  $s_0$  is added, the PA will be nondeterministic.

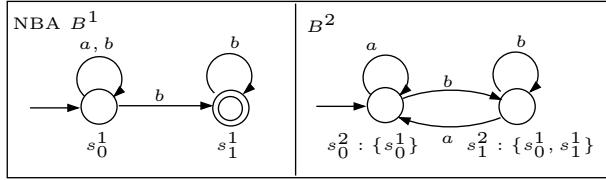
It can be directly observed that PA is an extension of Büchi automaton and Büchi automaton is in fact a special case of RA by designating  $S_R$  to be empty. More properties concerning partitioned automaton will be discussed in section 5.

## 4 Determinization of Büchi Automata

Basically, our transforming method is derived from the traditional subset construction for determinizing Nondeterministic Finite state Automata (NFA). Accordingly, to construct a Deterministic Partitioned Automaton (DPA)  $P = (S, \rightarrow, I, Q)$  of a nondeterministic Büchi automaton  $B^1 = (S^1, \rightarrow^1, I^1, F^1)$ , we first construct the deterministic version  $B^2 = (S^2, \rightarrow^2, I^2)$  of  $B^1$  by subset construction without designating the acceptable states. By subset construction, it is easily to be obtained that  $S^2 \subseteq 2^{S^1}$ , and  $I^2$  is a set with single member. Further, by  $B^1$  and  $B^2$ , the deterministic partitioned automaton  $P$  of  $B^1$  is constructed as follows. For convenience, we use  $s_0^1, s_1^1, s_2^1, \dots$  to denote the states in  $S^1$  and  $s_0^2, s_1^2, s_2^2, \dots$  to denote the states in  $S^2$ .

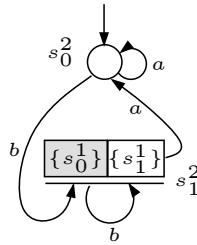
1. For any state  $s^2 \in S^2$ , if any  $s^1 \in s^2$  is non acceptable in  $B^1$ , i.e.  $s^1 \notin F^1$ , then  $s^2 \in S_C$ ; if any  $s^1 \in s^2$  is acceptable in  $B^1$ , i.e.  $s^1 \in F^1$ , then  $s^2 \in S_C$  and  $s^2 \in Q$ ; if  $s^2 = \{s_0^1, \dots, s_m^1, s_{m+1}^1, \dots, s_n^1\}$ , and for each  $0 \leq i \leq m$ ,  $s_i^1 \in F^1$ , while for each  $m < i \leq n$ ,  $s_i^1 \notin F^1$ , then  $s^2 \in S_R$  with  $\text{left}(s^2) = \{s_0^1, \dots, s_m^1\}$  and  $\text{right}(s^2) = \{s_{m+1}^1, \dots, s_n^1\}$ . Consequently,  $S = S_C \cup S_R \cup S_R^l \cup S_R^r$ .
2.  $I = I^2$ .
3. For each transition  $(s_i^2, a, s_{i+1}^2) \in \rightarrow^2$ ,
  - if  $s_i^2 \in S_C$ , then  $(s_i^2, a, s_{i+1}^2) \in \rightarrow$ ;
  - if  $s_i^2 \in S_R$ , and
    - if there exists  $s_k^1 \in \text{left}(s_i^2)$  and  $s_l^1 \in s_{i+1}^2$ , such that  $(s_k^1, a, s_l^1) \in \rightarrow^1$ , and
      - \* if there exists  $s_m^1 \in \text{right}(s_i^2)$  and  $s_n^1 \in s_{i+1}^2$  such that  $(s_m^1, a, s_n^1) \in \rightarrow^1$ , then  $(s_i^2, a, s_{i+1}^2) \in \rightarrow$ ;
      - \* if there exists no  $s_m^1 \in \text{right}(s_i^2)$  and  $s_n^1 \in s_{i+1}^2$  such that  $(s_n^1, a, s_m^1) \in \rightarrow^1$ , then  $(\text{left}(s_i^2), a, s_{i+1}^2) \in \rightarrow$ ;
    - if there exists  $s_k^1 \in \text{right}(s_i^2)$  and  $s_l^1 \in s_{i+1}^2$ , such that  $(s_k^1, a, s_l^1) \in \rightarrow^1$ , and there exists no  $s_m^1 \in \text{left}(s_i^2)$  and  $s_n^1 \in s_{i+1}^2$  such that  $(s_n^1, a, s_m^1) \in \rightarrow^1$ , then  $(\text{right}(s_i^2), a, s_{i+1}^2) \in \rightarrow$ .  $\square$

**Example 1.** Construct DPA  $P = (S, I, \rightarrow, Q)$  of the NBA  $B^1 = (S^1, I^1, \rightarrow^1, F^1)$  with  $S^1 = \{s_0^1, s_1^1\}$ ,  $I^1 = \{s_0^1\}$ ,  $\rightarrow^1 = \{(s_0^1, a, s_0^1), (s_0^1, b, s_0^1), (s_0^1, b, s_1^1), (s_1^1, b, s_1^1)\}$  and  $F^1 = \{s_1^1\}$  as depicted Fig.3.



**Fig. 3.** NBA  $B^1$ , Deterministic version  $B^2$  by subset construction

First,  $B^2 = (S^2, I^2, \rightarrow^2)$ , where  $S^2 = \{s_0^2, s_1^2\}$ ,  $I^2 = \{s_0^2\}$ , and  $\rightarrow^2 = \{(s_0^2, a, s_0^2), (s_0^2, b, s_1^2), (s_1^2, b, s_1^2), (s_1^2, a, s_0^2)\}$ , is obtained by applying subset construction to  $B^1$  without designating the accepting states as depicted in Fig.3. According to the transformation method, as illustrated in Fig.4,  $S_C = \{s_0^2\}$ ,  $S_R = \{s_1^2\}$  with  $left(s_1^2) = \{s_0^1\}$  and  $right(s_1^2) = \{s_1^1\}$ . Since both  $(s_0^1, b, s_0^1)$  and  $(s_1^1, b, s_1^1) \in \rightarrow^1$ ,  $(s_1^2, b, s_1^2) \in \rightarrow$ . Further,  $(right(s_1^2), a, s_0^2) \in$



**Fig. 4.** DPA  $P$

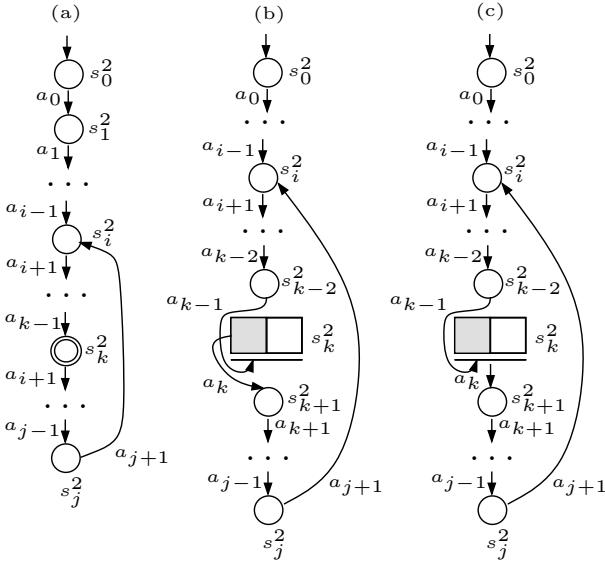
$\rightarrow$ , since  $(s_1^1, a, s_0^1) \in \rightarrow^1$  and there exist no transition labeled by  $a$  departing from  $s_0^1$  in  $\rightarrow^1$ .  $\square$

**Theorem 1.** For the deterministic partitioned automaton  $P = (S, \rightarrow, I, Q)$  constructed from a Büchi automaton  $B = (S^1, \rightarrow^1, I^1, F^1)$ ,  $L(P) = L(B)$ .

**Proof:** (1)  $L(P) \subseteq L(B)$ .

For an arbitrary word  $\xi \in L(P)$ , there must exist a unique run  $\rho$  of  $P$  on  $\xi$  such that  $\text{Inf}(\rho) \cap Q \neq \emptyset$ . According to the definition of the accepting set  $Q$ , there are three cases for  $\rho$ : (a) There exists at least one state  $s \in \text{Inf}(\rho)$  such that  $s \in Q$  and  $s \in S_C$ . (b) There exists at least one state  $s \in \text{Inf}(\rho)$  such that  $s \in Q$  and  $s \in S_R$ . (c) There exists at one state  $s \in \text{Inf}(\rho)$  such that  $s \in Q$  and  $s \in S_R^l$ .

For case (a), a run  $\rho$  of  $P$  on  $\xi$  can be illustrated as shown in Fig.5 (a). Accordingly, each state in the run is composed by a set of states in  $B$ . Suppose  $s_k^2 = \{s_0^1, \dots, s_n^1\}$  is acceptable and  $s_k^2 \subseteq F^1$ . By the construction of  $P$ , there must exist at least one infinite run  $\rho'$  in  $B$  on the infinite word  $\xi = a_0 a_1 a_{i-1} a_i, \dots, a_j, a_i, \dots, a_j, \dots$  such that  $\rho'(l) \in \rho(l)$ ,  $l \geq 0$ . Since,  $s_k^2 = \{s_0^1, \dots, s_n^1\}$  is a finite set, there must exist at least one state in  $s_k^2$ , let it be  $s_0^1$ , such that there are infinitely many  $m \in N_0$ , and  $\rho'(m) = s_0^1$ . Obviously,  $\rho'$  of  $B$  on  $\xi$  is acceptable in  $B$  since  $\text{Inf}(\rho') \cap F^1 = \{s_0^1\}$ . Thus  $\xi \in L(B)$ .

**Fig. 5.** Acceptable runs of  $P$  on  $\xi$ 

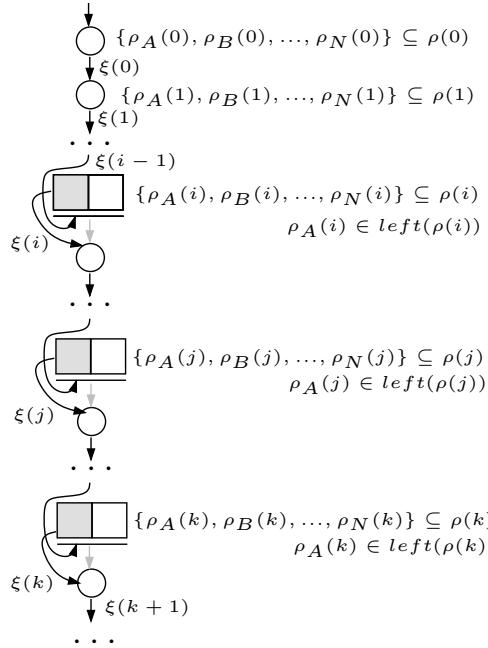
For case (b), a run  $\rho$  of  $P$  on  $\xi$  is depicted in Fig.5 (b). Suppose  $\text{left}(s_k^2) = \{s_0^1, \dots, s_n^1\}$  is acceptable and  $\text{left}(s_k^2) \subseteq F^1$ . By the construction of  $P$ , there must exist at least one infinite run  $\rho'$  in  $B$  on the infinite word  $\xi = a_0a_1a_{i-1}a_i, \dots, a_j, a_i, \dots, a_j, \dots$  such that  $\rho'(l) \in \rho(l)$ ,  $l \geq 0$ . Since,  $\text{left}(s_k^2) = \{s_0^1, \dots, s_n^1\}$  is a finite set, there must exist at least one state in  $s_k^2$ , let it be  $s_0^1$ , such that there are infinitely many  $m \in N_0$ , and  $\rho'(m) = s_0^1$ . Obviously,  $\rho'$  of  $B$  on  $\xi$  is acceptable in  $B$  since  $\text{Inf}(\rho') \cap F^1 = \{s_0^1\}$ . Thus  $\xi \in L(B)$ .

For case (c), a run  $\rho$  of  $P$  on  $\xi$  is depicted in Fig.5 (c). Suppose  $\text{left}(s_k^2) = \{s_0^1, \dots, s_n^1\}$  is acceptable and  $\text{left}(s_k^2) \subseteq F^1$ . By the construction of  $P$ , there must exist at least one infinite run  $\rho'$  in  $B$  on the infinite word  $\xi = a_0a_1a_{i-1}a_i, \dots, a_j, a_i, \dots, a_j, \dots$  such that  $\rho'(l) \in \rho(l)$ ,  $l \geq 0$  and  $l \neq k$ , while  $\rho(k) \in \text{left}(s_k^2)$ . Since,  $\text{left}(s_k^2) = \{s_0^1, \dots, s_n^1\}$  is a finite set, there must exist at least one state in  $s_k^2$ , let it be  $s_0^1$ , such that there are infinitely many  $m \in N_0$ , and  $\rho'(m) = s_0^1$ . Obviously,  $\rho'$  of  $B$  on  $\xi$  is acceptable in  $B$  since  $\text{Inf}(\rho') \cap F^1 = \{s_0^1\}$ . Thus  $\xi \in L(B)$ .

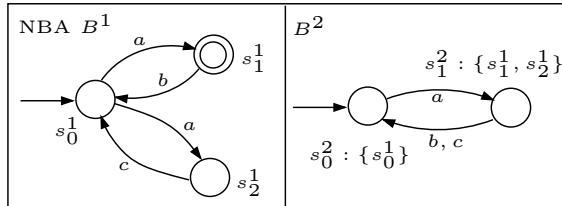
(2)  $L(B) \subseteq L(P)$ .

For a word  $\xi \in L(B)$ , there exists at least but may be more than one acceptable runs  $\rho_A, \rho_B, \dots, \rho_N$  of  $B$  on  $\xi$ . For convenience, without the loss of generality, let  $\rho_A(i), \rho_B(j), \dots, \rho_N(k) \in F^1$  are acceptable in  $B$ . By the construction of DPA from NBA, there must exist an unique acceptable run  $\rho$  of  $P$  on  $\xi$  where for each  $\rho(l)$ ,  $l \geq 0$ ,  $\{\rho_A(l), \rho_B(l), \dots, \rho_N(l)\} \subseteq \rho(l)$  as shown in Fig.6. Note that each transition departing from a left state may be replaced by a transition departing from the corresponding rectangle state (see the gray directed edges in Fig.6). By the definition for acceptable runs of DPA on infinite words,  $\rho$  of  $P$  on  $\xi$  is acceptable. Thus,  $\xi \in L(P)$ .  $\square$

Theorem 1 shows that any Büchi automata can be determined as a deterministic partitioned automaton via subset construction. As we known, subset construction is not enough for all the existing determinization construction. We present a counterexample



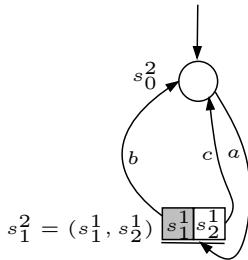
**Fig. 6.** An acceptable run of  $P$  on  $\xi$



**Fig. 7.** NBA  $B^1$ , Deterministic version  $B^2$  by subset construction

that illustrates that subset construction is insufficient to obtain an equivalent DPA from an NBA. As depicted in Fig. 7, a deterministic automaton  $B^2$  produced from NBA  $B^1$  by subset construction without designating the accepting set. Treated as a Muller automaton, the possible accepting table can be  $T = \{\{s_0^2\}, \{s_1^2\}, \{s_0^2, s_1^2\}\}$ . However,  $\{s_0^2\}$  and  $\{s_1^2\}$  are impossible to be accepting sets since any infinite run contains two states,  $s_0^2$  and  $s_1^2$ . So the only possible accepting set could be  $\{s_0^2, s_1^2\}$ , and we could have  $(a \cdot (b + c))^\omega \in L(B^2) = L(B^1)$ . This is not possible since  $B^1$  does not accept word  $(a \cdot c)^\omega$ . Therefore, it is impossible to designate Muller accepting table on  $B^2$  to equivalently accept the language defined by  $B^1$ .

By Theorem 1, partitioned automata is an omega automata where subset construction is sufficient for determinizing a nondeterministic Büchi automaton. For example, the NBA in Fig. 7 can be determinized as a DPA as depicted in Fig. 8.

**Fig. 8.** DPA of  $B^1$  in Figure 7

**Complexity Discussion:** By the construction of DPA from NBA, for an NBA  $B$  with  $n$  states, at most  $3 \times 2^n$  states will be created in the DPA  $P$ . Accordingly, the state complexity for determinizing NBA via partitioned automata is  $2^{O(n)}$ . However, in a DPA, different to the traditional automata, there are possibly three transitions between two states. Essentially, this brings that the transition complexity of partitioned automata will be  $3^n$  times larger than other traditional automata, such as Rabin, Street and Muller automata.

## 5 Properties of Partitioned Automaton

Being a new proposed omega automaton, properties, particularly, expressive powers of the deterministic and nondeterministic versions of PA are interesting researching points. From the definitions, it can be directly observed that PA is an extension of Büchi automaton and Büchi automaton is in fact a special case of RA by designating  $S_R$  to be empty. Further, by Theorem 1, the following corollary can be obtained.

**Corollary 2.** *Deterministic partitioned automaton has at least the same expressiveness with nondeterministic Büchi automaton.*  $\square$

Further, the following Lemma can be proved.

**Lemma 3.** *Any DPA can be equivalently transformed to a nondeterministic Büchi automaton.*

**Proof:** Given a DPA  $P = (S, \rightarrow, I, Q)$  with  $S = S_C \cup S_R \cup S_R^l \cup S_R^r$ . An NBA  $B = (S', \rightarrow', I', F)$  can be constructed as follows:

- $S' = S_C \cup S_R^l \cup S_R^r$ ;
- $I' = \begin{cases} I, & \text{if } I = \{s_0\} \text{ and } s_0 \in S_C \\ \{\text{left}(s_0), \text{right}(s)\}, & \text{if } I = \{s_0\} \text{ and } s_0 \in S_R \end{cases}$
- $F = Q \cap (S_C \cup S_R^l)$
- $\rightarrow' = \{(s, a, s') \mid (s, a, s') \in \rightarrow, s \in S_C \cup S_R^l \cup S_R^r \text{ and } s' \in S_C\}$   
 $\cup \{(s, a, \text{left}(s')), (s, a, \text{right}(s')) \mid (s, a, s') \in \rightarrow, s \in S_C \cup S_R^l \cup S_R^r \text{ and } s' \in S_R\}$   
 $\cup \{(\text{left}(s), a, s'), (\text{right}(s), a, s') \mid (s, a, s') \in \rightarrow, s \in S_R \text{ and } s' \in S_C\}$   
 $\cup \{(\text{left}(s), a, \text{left}(s')), (\text{right}(s), a, \text{left}(s')), (\text{left}(s), a, \text{right}(s')),$   
 $(\text{right}(s), a, \text{right}(s')) \mid (s, a, s') \in \rightarrow, s \in S_R \text{ and } s' \in S_R\}$

The proof for the correctness of the transformation is trivial and omitted here.  $\square$

Accordingly, Theorem 4 is ready to be proved.

**Theorem 4.** *Deterministic partitioned automaton has the same expressive power with nondeterministic Büchi automaton.*

**Proof:** This theorem is a direct consequence of Theorem 1 and Lemma 3.  $\square$

Similar to Lemma 3, Lemma 5 for nondeterministic partitioned automaton can also be proved.

**Lemma 5.** *Any partitioned automaton can be equivalently transformed to a nondeterministic Büchi automaton.*

**Proof:** The lemma can be proved by a construction from PA to NBA similar to the one from DPA to NBA.  $\square$

**Theorem 6.** *Any partitioned automaton can be determinized as a deterministic partitioned automaton.*

**Proof:** This theorem is a direct consequence of Theorem 1 and Lemma 5.  $\square$

Moreover, since DPA is a special case of PA, the following corollary is obtained.

**Corollary 7.** *Nondeterministic partitioned automaton, deterministic partitioned automaton and nondeterministic Büchi automaton have the same expressive power.*  $\square$

## 6 Conclusions

By the introduction of partitioned automaton, for which both the deterministic and nondeterministic versions have the same expressive power with NBA, a new determinization construction of Büchi automata is proposed. For partitioned automaton is essentially different to the traditional automata, state spaces for determinizing Büchi automata are exponentially saved by considerable waste of transitions. Intuitively, the new construction is easily implementable since it is completely based on subset construction. In the near future, we will further study whether deterministic partitioned automata are useful in automata based verification and complementing Büchi automata.

## References

1. Clark, M., Gremberg, O., Peled, A.: Model Checking. The MIT Press (2000)
2. Katoen, J.-P.: Concepts, Algorithms and Tools for Model Checking. Arbeitsberichte der Informatik, Friedrich-Alexander-Universitaet Erlangen-Nuernberg 32(1) (1999)
3. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Proceedings of the International Congress on Logic, Method, and Philosophy of Science, pp. 1–12. Stanford University Press (1962)
4. Thomas, W.: Languages, automata and logic. In: Handbook of Formal Languages, vol. 3, pp. 389–455. Springer, Berlin (1997)

5. Holzmann, G.J.: The Model Checker Spin. *IEEE Trans. on Software Engineering* 23(5), 279–295 (1997)
6. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games. LNCS, vol. 2500. Springer, Heidelberg (2002)
7. Jain, H.: Automata on Infinite Objects. B.Tech Seminar Report, India, internal report (1996)
8. Vardi, M.Y.: The Büchi Complementation Saga. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 12–22. Springer, Heidelberg (2007)
9. Safra, S.: On the complexity of omega-automata. In: Proceedings of the 29th Annual Symposium on Foundations of Computer Science, FOCS 1988, pp. 319–327. IEEE Computer Society Press (1988)
10. Muller, D.E., Schupp: Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science* 141(1-2), 69–107 (1995)
11. Boker, U., Kupferman, O.: Co-ing Büchi Made Tight and Useful. In: LICS 2009, pp. 245–254 (2009)
12. Boker, U., Kupferman, O.: The Quest for a Tight Translation of Büchi to co-Büchi Automata. In: Blass, A., Dershowitz, N., Reisig, W. (eds.) Fields of Logic and Computation. LNCS, vol. 6300, pp. 147–164. Springer, Heidelberg (2010)
13. Boker, U., Kupferman, O.: Co-Büching Them All. In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 184–198. Springer, Heidelberg (2011)
14. Friedgut, E., Kupferman, O., Vardi, M.Y.: Büchi complementation made tighter. *International Journal of Foundations of Computer Science* 17(4), 851–863 (2006)
15. Sistla, A.P., Vardi, M.Y., Wolper, P.: The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science* 49, 217–237 (1987)
16. Muller, D.E.: Infinite sequences and finite machines. In: Proceedings of the 4th IEEE Symposium on Switching Circuit Theory and Logical Design, pp. 3–16 (1963)
17. McNaughton: Testing and generating infinite sequences by a finite automaton. *Information and Control* 9(5), 521–530 (1966)
18. Yan, Q.: Lower bounds for complementation of  $\omega$ -automata via the full automata technique. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 589–600. Springer, Heidelberg (2006)
19. Schewe, S.: Büchi Complementation Made Tight. In: STACS 2009, pp. 661–672 (2009)
20. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. *Journal of Logical Methods in Computer Science* 3 (2007)
21. Liu, W., Wang, J.: A tighter analysis of Piterman’s Büchi determinization. *Inf. Process. Lett.* 109(16), 941–945 (2009)
22. Schewe, S.: Tighter Bounds for the Determinisation of Büchi Automata. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 167–181. Springer, Heidelberg (2009)
23. Colcombet, T., Zdanowski, K.: A Tight Lower Bound for Determinization of Transition Labelled Büchi Automata. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part II. LNCS, vol. 5556, pp. 151–162. Springer, Heidelberg (2009)

# On Linear-Size Pseudorandom Generators and Hardcore Functions

Joshua Baron<sup>1</sup>, Yuval Ishai<sup>2</sup>, and Rafail Ostrovsky<sup>3</sup>

<sup>1</sup> HRL Laboratories, Malibu, CA, USA 90265

jwbaron@hrl.com

<sup>2</sup> Technion, Haifa, Israel 32000

yuvali@cs.technion.ac.il

<sup>3</sup> UCLA, Los Angeles, CA, USA 90095

rafail@cs.ucla.edu

**Abstract.** We consider the question of constructing pseudorandom generators that simultaneously have linear circuit complexity (in the output length), exponential security (in the seed length), and a large stretch (linear or polynomial in the seed length). We refer to such a pseudorandom generator as an *asymptotically optimal PRG*. We present a simple construction of an asymptotically optimal PRG from any one-way function  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  which satisfies the following requirements:

1.  $f$  can be computed by linear-size circuits;
2.  $f$  is  $2^{\beta n}$ -hard to invert for some constant  $\beta > 0$ , and the min-entropy of  $f(x)$  on a random input  $x$  is at least  $\gamma n$  for a constant  $\gamma > 0$  such that  $\beta/3 + \gamma > 1$ .

Alternatively, building on the work of Haitner, Harnik and Reingold (SICOMP 2011), one can replace the second requirement by:

- 2'.  $f$  is  $2^{\beta n}$ -hard to invert for some constant  $\beta > 0$  and it is *regular* in the sense that the preimage size of every output of  $f$  is fixed (but possibly unknown).

Previous constructions of PRGs from one-way functions can do without the entropy or regularity requirements, but even the best such constructions achieve slightly sub-exponential security (Vadhan and Zheng, STOC 2012).

Our construction relies on a technical result about hardcore functions that may be of independent interest. We obtain a family of hardcore functions  $\mathcal{H} = \{h : \{0, 1\}^n \rightarrow \{0, 1\}^{\alpha n}\}$  that can be computed by linear-sized circuits for any  $2^{\beta n}$ -hard one-way function  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  where  $\beta > 3\alpha$ . Our construction of asymptotically optimal PRGs uses such hardcore functions, which can be obtained via linear-size computable affine hash functions (Ishai, Kushilevitz, Ostrovsky and Sahai, STOC 2008).

**Keywords:** Pseudorandom generators, hardcore functions, circuit complexity, exponential hardness, pairwise independence, bilinear hash families.

## 1 Introduction

A *pseudorandom generator* (PRG) [7,30] is a deterministic algorithm which stretches a short random seed into a longer output which looks random to any computationally bounded observer. PRGs have numerous applications in cryptography. In particular, they serve as useful building blocks for basic cryptographic tasks such as (symmetric) encryption, commitment, and message authentication.

A seemingly weaker primitive, which satisfies a much milder form of hardness requirement, is a *one-way function* (OWF). A OWF is an efficiently computable function which is hard to invert on a random input. We say that  $f$  is  $t(n)$ -hard to invert (or  $t(n)$ -hard for short) if every algorithm running in time  $t(n)$  can find a preimage of  $f(x)$  for a random  $x \in \{0, 1\}^n$  with at most  $1/t(n)$  probability, for all sufficiently large  $n$ . We say that  $f$  is *exponentially hard* if it is  $2^{\beta n}$ -hard for some constant  $\beta > 0$ .

Every PRG which significantly stretches its seed is also a OWF. However, because of its crude form of security, a OWF is easier to construct heuristically than a PRG. There are many natural candidates for a OWF (even an exponentially strong OWF) which do not immediately give rise to a similar PRG. This motivated a line of work on constructing PRGs from different types of OWFs, which culminated in the seminal result of Håstad, Impagliazzo, Levin and Luby (HILL) [21] that a PRG can be constructed from an *arbitrary* OWF. More recently, there has been another fruitful line of work on simplifying and improving the efficiency of the HILL construction [22,17,18,19,20,16,29].

The main focus in the above works has been on optimizing efficiency under *minimal assumptions*. The present work is motivated by the following dual question: under which assumptions can we obtain *optimal efficiency*? Ideally, we would like to obtain a PRG  $G : \{0, 1\}^n \rightarrow \{0, 1\}^{l(n)}$  satisfying the following requirements:

- $G$  has *large stretch*; that is,  $l(n) > cn$  or even  $l(n) > n^c$  for some constant  $c > 1$ . A large stretch is crucial for most cryptographic applications of PRGs.
- $G$  has *linear circuit complexity*; that is, the output of  $G$  can be computed by a uniform family of (bounded fan-in) boolean circuits of size  $O(l(n))$ . This implies linear-time computation also in other, more liberal, models such as unbounded fan-in circuits or different flavors of RAM.
- $G$  has *exponential security*; that is, there exists a constant  $\delta > 0$  such that any algorithm running in time  $2^{\delta n}$  can distinguish between the output of  $G$  and a truly random string of length  $l(n)$  with at most a  $2^{-\delta n}$  advantage. In typical PRG applications, exponential security is useful for minimizing the asymptotic length of the secret keys or the amount of true randomness.

We refer to a PRG as above as an *asymptotically optimal PRG*. Using this terminology, the main question we pose in this work is the following:

*Which types of one-way functions imply an asymptotically optimal PRG?*

The above question is motivated by the broad goal of obtaining efficient cryptographic constructions whose security can be proved under conservative assumptions. Indeed, the efficiency of encryption schemes and other cryptographic applications of PRGs is often dominated by the efficiency of the underlying PRG [25].

A natural conjecture is that an asymptotically optimal PRG can be constructed from any OWF  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  which can be computed by linear-size circuits and is exponentially hard to invert. However, this conjecture does not seem to follow from the current state of the art. A recent result of Vadhan and Zheng [29] (improving on [18,20]) comes close to proving the conjecture. Combined with linear-size computable pairwise independent hash functions [25], the result from [29] implies a PRG construction which satisfies the first two requirements but falls short of the third. More concretely, the construction adds a  $\text{polylog}(n)$  multiplicative overhead to the seed length.

A recent PRG construction of Applebaum [3] satisfies the first two requirements and has the additional feature of a constant output locality (namely, each output bit depends on a constant number of input bits). This construction relies on variants of a one-wayness assumption due to Goldreich [12]. Roughly speaking, this assumption asserts that a randomly chosen function from the class of functions having constant output locality is one-way with high probability.

A construction of an asymptotically optimal PRG based on an exponential version of an indistinguishability assumption due to Alekhnovich [1] follows from the work of Applebaum, Ishai, and Kushilevitz [6] (see also [25,3]). The question of constructing asymptotically optimal PRGs under more general assumptions remained open.

## 1.1 Our Contribution

We prove the above conjecture for one-way functions  $f$  that are either “regular” (in the sense that every output  $f(x)$  has the same number of preimages) or alternatively have a “random enough output” on a random input  $x$ . More concretely, we prove the following result:

**Theorem 1 (Asymptotically Optimal PRGs).** *Suppose that  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  is  $2^{\beta n}$ -hard to invert for some  $\beta > 0$ . Suppose that either  $f$  is regular or the min-entropy of  $f(x)$  is larger than  $\gamma n$  for some constant  $\gamma$  such that  $\gamma > 1 - \beta/3$  (and for sufficiently large  $n$ ). Then there exists an exponentially strong PRG  $G : \{0, 1\}^n \rightarrow \{0, 1\}^{2n}$  that can be computed by linear-size circuits using  $O(1)$  oracle calls to  $f$ .*

Using a standard tree-based PRG extension, the above theorem yields an asymptotically optimal PRG with an arbitrary polynomial stretch; see the full version of this paper for further details.

The above entropy requirement seems quite mild and in some cases of interest it can be proved unconditionally. In particular, there are natural variants of Goldreich’s OWF candidate [12] that can be shown to have fractional entropy

that tends to 1 with the locality degree (see [9] and the full version of this paper), whereas the expected hardness of inverting does not seem to decrease (and may even grow) with the locality.

**Hardcore Functions.** Our construction of asymptotically optimal PRGs is obtained via a technical result about hardcore functions that may be of independent interest. Recall that a hardcore predicate  $b$  is a function that outputs a single bit  $b(x)$  which is hard to predict even given  $f(x)$ . A hardcore *function* for a one-way function  $f$  is a function  $h$  (which can output more than one bit) whose output  $h(x)$  is hard to distinguish from random even when  $f(x)$  is known. More precisely, we allow  $h$  to be picked at random from a function family  $H$  and provide a description of  $h$  as an additional input to the distinguisher. Hardcore functions are a fundamental cryptographic object, with applications to pseudoentropy and pseudorandomness. Goldreich and Levin [15] introduced the first hardcore predicates and functions for general OWFs, showing that a random linear function is hardcore and so is the linear function defined by a random Toeplitz matrix.

We consider families of linear functions  $H = \{h_i : \{0, 1\}^n \rightarrow \{0, 1\}^m\}$  over the binary field. We refer to such a family as a *bilinear uniform output hash family* if it satisfies two properties. First, for any  $x \neq 0$ , the random variable  $h_i(x)$  (induced by a uniformly random choice of the index  $i$ ) is uniformly distributed over  $\{0, 1\}^m$ . Second,  $H$  forms a subgroup of the (additive) group of linear functions from  $\mathbb{F}_2^n$  to  $\mathbb{F}_2^m$ . Using a result of Holenstein, Maurer, and Sjödin [23], we show that any such family of functions is hardcore for any sufficiently hard OWF.

**Theorem 2 (Bilinear Uniform-Output Hash Families are Hardcore).** *Let  $H = \{h : \{0, 1\}^n \rightarrow \{0, 1\}^{\alpha n}\}$  be a bilinear uniform output hash family and let  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  be a  $2^{\beta n}$ -hard one-way function. Then  $H$  is a family of exponentially strong hardcore functions for  $f$  if  $\beta > 3\alpha$ .*

A construction of linear-size computable pairwise independent hash functions was given by Ishai, Kushilevitz, Ostrovsky and Sahai [25]. Observing that the construction can be instantiated so that each function in the family is affine, and constructing linear uniform-output hash families from such families, we can use the above theorem to obtain linear-size computable hardcore functions with a long output. Using such a hardcore function, the construction of an asymptotically optimal PRG proceeds in a simple way. In the high entropy case, we first extract the randomness from the output of  $f$  by applying a (linear-size) pairwise independent hash function (appealing to the Leftover Hash Lemma [21]). Then, we extract sufficient pseudorandomness from the input of  $f$  by applying the (linear-size) hardcore function to the input  $x$ . If  $f$  has sufficiently high min-entropy and is hard enough to invert, these techniques combine so that the output has length  $cn$  for  $c > 1$ . From this PRG, we can use standard PRG extension techniques to obtain an asymptotically optimal PRG with an arbitrary polynomial stretch.

In the case where the OWF  $f$  is regular, we combine the hardcore function result with a PRG construction of Haitner, Harnik and Reingold [17,19] (the

HILL construction [21] yields a similar result for regular  $f$  with known preimage size).

## 1.2 Related Work

**Pseudorandom Generators.** Following the pioneering works of Blum and Micali [7] and Yao [30], who constructed a PRG from a one-way *permutation*, Goldreich, Krawczyk and Luby [14] constructed a PRG from any *regular* OWF with unknown preimage size (a OWF is regular if every output of  $f$  has the same preimage size). Håstad, Impagliazzo, Levin and Luby [21] gave the first construction of a PRG from any OWF. The first effort towards simplifying and improving the HILL construction was made by Holenstein [22], who also explicitly considered the case of exponentially strong OWFs. Haitner, Harnik and Reingold [17,18,19] improved the construction of [14] by relying only on pairwise independent hash functions ([14] had required  $n$ -wise independent hash functions) and by reducing the seed length. More recently, Haitner, Reingold and Vadhan [20] further improved the seed length of PRGs from general OWFs. The most efficient general constructions to date are given in the aforementioned work of Vadhan and Zheng [29], who also noted that combining their construction with the pairwise independent hash functions of [25] gives a linear-stretch linear-size PRG from *any* exponentially hard OWF. (This construction does not depend on the hash functions being affine.) As discussed above, this construction still falls short of our main goal because of its polylogarithmic overhead to the seed length, but otherwise it is stronger in several important aspects. In particular, it does not require  $f$  to satisfy any entropy or regularity requirement.

Constructions of PRGs in  $\text{NC}^0$  (i.e., with constant output locality) were first given by Applebaum, Ishai, and Kushilevitz [5] under standard assumptions. Note that any  $\text{NC}^0$  function can be realized by linear-size circuits (in the output length). However, the PRGs in  $\text{NC}^0$  from [5] have sublinear stretch. Linear-stretch PRGs in  $\text{NC}^0$  were constructed in [6] under an *indistinguishability* assumption due to Alekhnovich [1]. Under an exponentially strong version of the assumption from [1], this construction yields an asymptotically optimal PRG.

A family of linear-stretch PRGs in  $\text{NC}^0$  whose security is based on a natural *one-wayness* assumption was given by Applebaum [3], who under similar assumptions also obtained a PRG with polynomial stretch in  $\text{NC}^0$ . However, the security level of the PRGs constructed in [3] does not meet the third requirement of an asymptotically optimal PRG, even under exponential one-wayness assumptions. Furthermore, the underlying OWFs in these constructions are restricted to special distributions over  $\text{NC}^0$  functions, whereas our construction does not require the underlying OWF to be in  $\text{NC}^0$  (nor does it yield a PRG in  $\text{NC}^0$ ).

Finally, Applebaum, Bogdanov, and Rosen [4] (following earlier works of Cryan and Miltersen [11] and Mossel, Shpilka, and Trevisan [26]) present a broad class of randomized constructions of small-bias PRGs in  $\text{NC}^0$ , namely PRGs in  $\text{NC}^0$  which provably fool all *linear* distinguishers. Such small-bias PRGs may serve as plausible candidates for asymptotically optimal PRGs, though their security does not seem to follow from any natural one-wayness assumption.

**Goldreich’s One-Way Function.** Goldreich [12] put forward the following graph-based one-way function candidate: Consider a  $d$ -ary (nonlinear) predicate  $P$  and a bipartite graph  $G = (V, E)$  with left nodes  $u_1, \dots, u_n$ , right nodes  $v_1, \dots, v_n$ , and right degree  $d$ . Define  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  by labeling the left (input) nodes of  $G$  with the bits of  $x$ . We define the  $j$ th output bit of  $f$  on  $x$ ,  $f(x)_j$ , as  $P(x_{i_1}, \dots, x_{i_d})$ , where  $u_{i_1}, \dots, u_{i_d}$  are the input nodes that are in the neighborhood of  $v_j$ , the  $j$ th output node of  $G$ . We note that  $f$  can be computed by linear-size circuits as long as  $d$  is a constant. Goldreich conjectured that most functions  $f$  as above are one-way.

Recent works on this class of functions [28, 2, 9, 10, 8] may be viewed as supporting the possibility that they are exponentially hard; however, Bogdanov and Qiao [8] have shown that for variants where the output stretch is a large constant (at least exponential in the input degree), there exist instantiations that are invertible in polynomial time. Applebaum’s construction of a linear-stretch PRG in  $\text{NC}^0$  [3] uses a variant of Goldreich’s one-way function with a large constant stretch. He demonstrates that the one-wayness of such local functions implies that the output has sufficiently good pseudoentropy to allow the construction of a PRG. By contrast, the one-way functions required for the constructions in this paper are from  $n$  bits to  $n$  bits and, as discussed above, the security reduction from [3] is not tight enough to yield an asymptotically optimal PRG.

In the full version of this paper, we show that a random  $d$ -local one-way function from  $n$  bits to  $n$  bits, instantiated with a random and independent  $d$ -ary predicate for *each* output bit of the function, has high min-entropy except with exponentially small probability over the choice of graph and predicates. This is useful towards instantiating the types of OWFs on which our main result relies. Previous works (e.g., [9, 10]) have examined instances with more concrete choices of the predicate  $P$  and proved them also to have high min-entropy except with exponentially small probability over the choice of the function.

**Hardcore Functions.** Goldreich and Levin [15] demonstrated that the set of all inner product functions constitutes a family of hardcore predicates for any one-way function. More generally, they proved that the set of all linear functions with input in  $\{0, 1\}^n$  and the set of Toeplitz matrices with input  $\{0, 1\}^n$  are families of hardcore functions for any one-way function (for appropriately sized outputs). The central idea of their proof is that if a random XOR of a candidate hardcore function output is hard to distinguish, then the function is indeed hardcore; they constructed such an argument for the set of all matrices and Toeplitz matrices, respectively, by direct calculation.

Näslund [27] showed that the family of all affine functions over  $GF[2^n]$  and the family of all linear functions over the integers modulo a prime are families of hardcore functions for any one-way function.

Holenstein, Maurer and Sjödin [23] generalized the results of [15] to give a complete classification of all so-called bilinear hardcore function families over arbitrary fields; that is, the hardcore functions are additively homomorphic both in their function inputs and in the strings that represent each function (this

is the case when the set of hardcore functions forms an additive group). Our construction of linear-size computable hardcore functions will rely on this result.

### 1.3 Definitions and Preliminaries

We denote by  $U_n$  the random variable uniformly distributed over  $\{0, 1\}^n$ . We provide some definitions and preliminaries used in this paper; see the full version for other definitions, such as bilinear functions, full rank bilinear functions, one-way functions, hardcore functions, pseudorandom generators and pairwise independent hash families. In particular, we say that a function is a  *$\beta$ -exponential one-way function* if it is a  $(2^{\beta n}, 2^{-\beta n})$  one-way function. We also say that a pairwise independent hash family where each function in the family is affine is an *affine pairwise independent (API) hash family*.

**Definition 3.** Let  $H_{n,m} = \{h_i : \{0, 1\}^n \rightarrow \{0, 1\}^m\}$  be a multiset of functions (that is, we allow distinct indices to represent the same function). We say that  $H_{n,m}$  is a family of uniform-output hash functions if for every non-zero  $x \in \{0, 1\}^n$ , the random variables  $H_{n,m}(x)$  induced by a uniform choice of  $h$  from  $H_{n,m}$  is uniformly distributed over  $\{0, 1\}^m$ . If every  $h_i \in H_{n,m}$  is a linear function over the binary field (i.e., a function of the form  $A_i x$ ), we call  $H_{n,m}$  a linear uniform-output (LUO) hash family.

Further, a LUO hash family of size  $2^k$  that can be expressed as a bilinear function  $h : \{0, 1\}^n \times \{0, 1\}^k \rightarrow \{0, 1\}^m$  (where the second argument represents the index  $i$ ) is denoted a bilinear uniform-output (BLUO) hash family.

We will typically consider infinite collections of families  $H_{n,m}$  parameterized by the input and output length. In such a case we require the existence of a representation length  $\ell_{n,m} = \text{poly}(n, m)$  such that  $H_{n,m}$  contains  $2^{\ell_{n,m}}$  (not necessarily distinct) functions  $h_i$  indexed by all binary strings of length  $\ell_{n,m}$  (which equals  $k$  in the above definitions). For convenience, we will abuse notation and refer to  $h_i \in H_{n,m}$  as both a function and the string representing it. We assume that there is a polynomial-time evaluation algorithm that, given  $h_i$  and  $x$ , outputs  $h_i(x)$ . In fact, we will rely on families for which this algorithm can be implemented by linear-size circuits.

**Claim 4.** Let  $H'_{n,m}$  be an API hash family. Then the multiset  $H_{n,m} = \{h_i : h_i(x) = h'_i(x) - h'_i(0), h'_i \in H'\}$  is an LUO hash family.

The proof of the claim is immediate from the fact that for any  $x \neq 0$ ,  $H'_{n,m}(x)$  and  $H'_{n,m}(0)$  are distributed uniformly and independently at random, and that each function in  $H_{n,m}$  is linear.

## 2 Linear-Size Hardcore Functions

We now give our main result for the existence of linear-size hardcore functions.

**Theorem 5.** Let  $H_{n,l(n)}$  be a BLUO hash family, let  $f : \{0,1\}^n \rightarrow \{0,1\}^n$  be a  $(t(n), 1/t(n))$  one-way function, and let  $\theta > 1$  be an arbitrary constant. Then  $H_{n,l(n)}$  is a  $(t'(n), 1/t'(n))$  family of hardcore functions for  $f$  if  $3\theta(l(n) + \log t'(n)) < \log t(n)$ .

**Corollary 6.** Let  $H_{n,l(n)}$  be a BLUO hash family and let  $f : \{0,1\}^n \rightarrow \{0,1\}^n$  be a one-way (resp.  $\beta$ -exponential one-way) function. Then  $H_{n,l(n)}$  is a family of hardcore functions (resp. is a  $(2^{\Omega(n)}, 2^{-\Omega(n)})$  family of hardcore functions) for  $f$  for any  $l(n) \in O(\log n)$  (resp.  $l(n) < \frac{\beta n}{3\theta}$  for any constant  $\theta > 1$ ).

We initiate the proof of Theorem 5 by proving a technical lemma.

**Lemma 7.** Let  $\mathcal{H}$  be a BLUO hash family specified by the bilinear function  $h$ . Then  $h$  is full rank.

**Proof of Lemma 7.** Let  $h : \{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^m$  be the bilinear function that specifies  $\mathcal{H}$ . Then, by definition of LUO hash families, for any  $0 \neq x \in \{0,1\}^n$ , the distribution  $\{h(x, r)\}_{r \leftarrow U_k}$  is distributed uniformly over  $\{0,1\}^m$ . Let  $l : \{0,1\}^m \rightarrow \{0,1\}$  be an arbitrary non-zero linear function. Then for any  $0 \neq x \in \{0,1\}^n$ ,  $\{l \circ h(x, r)\}_{r \leftarrow U_k}$  is also distributed uniformly over  $\{0,1\}$  and, in particular, the linear map  $r \mapsto h(x, r)$  is surjective onto  $\{0,1\}$  for any non-zero  $x$ .

Let  $M_l$  be the  $n \times k$  matrix denoting  $l \circ h : \{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}$ . We would like to prove that  $\text{rank}(M_l) = n$ . This follows from the fact that for every non-zero  $x \in \{0,1\}^n$ , there exists some  $r_x$  such that  $x^T \cdot M \cdot r_x = 1$ , which implies that every non-trivial linear combination of the rows of  $M_l$  is non-zero, and the lemma follows.  $\square$

We now proceed to prove Theorem 5.

**Proof of Theorem 5.** We will proceed by contradiction, and assume that there exists a probabilistic algorithm  $D$  running in time  $t'(n)$  such that  $|\Pr[x \leftarrow U_n, h \leftarrow H_{n,l(n)} : D(f(x), h, h(x)) = 1] - \Pr[x \leftarrow U_n, h \leftarrow H_{n,l(n)}, y \leftarrow U_{l(n)} : D(f(x), h, y) = 1]| > \epsilon'(n) = 1/t'(n)$  for infinitely many  $n$ .

More specifically, let  $\Pr[x \leftarrow U_n, h \leftarrow H_{n,l(n)} : D(f(x), h, h(x)) = 1] = \delta$  and  $\Pr[x \leftarrow U_n, h \leftarrow H_{n,l(n)}, y \leftarrow U_{l(n)} : D(f(x), h, y) = 1] = (1 + \epsilon)\delta$ . Without loss of generality, let  $\epsilon > 0$  (otherwise, let  $D'$  be the algorithm that outputs the opposite bit that  $D$  does and use  $D'$  for the remainder of this proof); then  $\epsilon\delta \geq \epsilon'(n)$ .

Let  $\alpha(n)$  be such that for some  $\theta > 1$ ,  $3\theta(l(n) + \log t'(n)) + \log \alpha(n) < \log t(n)$ . Since by Lemma 7,  $\mathcal{H}$  can be specified by a full rank bilinear function, a slightly modified version of the hardcore result of [23] (see the full version for details) implies that there exists an algorithm  $A_\alpha$  and some  $c > 0$  that inverts  $f$  in time  $\alpha(n) \cdot \frac{2^{2l(n)}}{\delta\epsilon^2} \cdot n^c \cdot t'(n) \leq \alpha(n) \cdot \frac{2^{2l(n)+1}}{\epsilon'(n)^2} \cdot n^c \cdot t'(n) = \alpha(n) \cdot 2^{2l(n)+1} \cdot n^c \cdot t'^3(n)$ , which, by assumption, is less than  $t(n)$ . Further,  $A_\alpha$  inverts  $f$  with probability  $\geq \frac{\delta\epsilon^2}{4 \cdot 2^{2l(n)}} - \frac{1}{\alpha(n)} \geq \frac{\epsilon'(n)^2}{4 \cdot 2^{2l(n)}} - \frac{1}{\alpha(n)}$ , which, by assumption, is larger than  $\epsilon(n) = 1/t(n)$ , contradicting the one-wayness of  $f$ .  $\square$

Ishai et al [25] construct a family of pairwise independent hash functions from  $\{0, 1\}^n$  to  $\{0, 1\}^n$  which can be computed by circuits of size  $O(n)$ . This construction uses an arbitrary pairwise independent hash function (applied on a constant-size input domain) as a building block. Using a family of *affine* functions as a building block (e.g.,  $Ax + b$  where  $A$  is a random binary matrix and  $b$  a random binary vector) yields a linear-size computable family of *affine* pairwise independent hash functions. We use this family to construct a linear-size computable BLUO hash family<sup>1</sup>.

**Proposition 8 (Implicit in [25]).** *For any  $0 < c \leq 1$ , there exists a family of affine pairwise independent hash functions from  $\{0, 1\}^n$  to  $\{0, 1\}^{cn}$  which can be computed by linear-size circuits.*

Combining Proposition 8 with Claim 4, we obtain the following lemma.

**Lemma 9.** *For any  $0 < c \leq 1$ , there exists a BLUO hash family from  $\{0, 1\}^n$  to  $\{0, 1\}^{cn}$  which can be computed by linear-size circuits.*

Applying Theorem 5 with these hash functions yields the following result.

**Corollary 10.** *Let  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  be a  $\beta$ -exponential one-way function. For any  $l(n)$  and  $\theta > 1$  such that  $3l(n) < \beta n\theta$ , there exists a BLUO hash family  $H_{n,l(n)}$  such that  $H_{n,l(n)}$  is a  $(2^{\Omega(n)}, 2^{-\Omega(n)})$  family of hardcore functions of  $f$  which can be computed by linear-size circuits.*

### 3 PRGs Computable by Linear-Size Circuits

We discuss how hardcore functions that can be computed by linear-size circuits can be used to construct linear-stretch PRGs that can be computed by linear-size circuits. When  $f$  is an exponentially hard one-way function, various assumptions about the min-entropy of the output of  $f$  can be used to construct such PRGs. We first construct a PRG in the case that the output of  $f$  has high enough min-entropy. We then examine previously made restrictions on  $f$  that have been used to construct linear-stretch PRGs.

#### 3.1 PRGs for One-Way Functions with Lower-Bounded Min-Entropy

We demonstrate that there exist linear-stretch pseudorandom number generators that can be computed by linear-size circuits provided that there exists a suitable class of exponentially hard one-way functions. We discuss the plausibility of these assumptions in the full version of this paper.

---

<sup>1</sup> When we say that a family of hash functions can be computed by linear-size circuits we mean that there is a universal constant  $c$  such that for every sufficiently large  $m$  and  $n$ , there is a circuit  $C_{n,m}$  of size  $cn$  which computes the restriction of the family to functions from  $\{0, 1\}^n$  to  $\{0, 1\}^m$ . The input to  $C_{n,m}$  includes the (binary representation of) the index  $i$  of the hash function and the input for  $h_i$ .

**Assumption 11.** *There exist a function  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  and constants  $\beta > 0, \theta > 1$ , and  $\gamma$  such that  $\gamma > 1 - \frac{\beta}{3\theta}$  with the following properties:*

- (i)  *$f$  is a  $\beta$ -exponential one-way function.*
- (ii)  *$f$  can be computed by linear-size circuits.*
- (iii)  *$H_\infty(f(U_n)) > \gamma n$ .*

Under this assumption, the following theorem can be proved.

**Theorem 12.** *If Assumption 11 holds, there exists a  $(2^{\Omega(n)}, 2^{-\Omega(n)})$  linear-stretch PRG  $G$  that can be computed by linear-size circuits with a single oracle call to  $f$ .*

Using either the construction in [25] or the construction in [13] (see Section 3.3.2 there) with the PRG  $G$  of Theorem 12, we obtain the following corollary.

**Corollary 13.** *If Assumption 11 holds, then for any polynomial  $l(n) > n$  there exists a  $(2^{\Omega(n)}, 2^{-\Omega(n)})$  PRG  $G : \{0, 1\}^n \rightarrow \{0, 1\}^{l(n)}$  such that  $G$  can be computed by circuits of size  $O(l(n))$  with  $O(l(n)/n)$  oracle calls to  $f$ .*

In the following Construction 14, we describe an algorithm that we prove satisfies Theorem 12; see the full version for the full proof of Theorem 12. It can be shown that it is possible to specify a BLUO hash family (and also an API hash family)  $h \in H_{m, \alpha m}$  by specifying a string from  $\{0, 1\}^{\mu m}$  for some constant  $\mu$ . Let  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  be a  $\beta$ -exponential one-way function. Set  $c_0 = \gamma$  and  $c_1 = 1 - \gamma + \epsilon_2$  for any constant  $0 < \epsilon_2 < \gamma - (1 - \frac{\beta}{3\theta})$ . Construct an API hash family,  $H_{n, c_0 n}$ , and a BLUO hash family,  $H_{n, c_1 n}$ , which are indexed by the sets  $\{0, 1\}^{k_0 n}$  and  $\{0, 1\}^{k_1 n}$  for some constants  $k_0$  and  $k_1$ , respectively.

**Construction 14.** *Let  $H_{n, c_0 n}$  be an API hash family and  $H_{n, c_1 n}$  be a BLUO hash family with  $h_0$  and  $h_1$  drawn from  $H_{n, c_0 n}$  and  $H_{n, c_1 n}$ , respectively. Let  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  satisfy Assumption 11. Then set:*

$$G(x, h_0, h_1) = (h_0, h_0(f(x)), h_1, h_1(x)).$$

Note that  $|(x, h_0, h_1)| = (1 + k_0 + k_1)n$  and  $|G(x, h_0, h_1)| = k_0 n + c_0 n + k_1 n + c_1 n = (1 + \epsilon_2 + k_0 + k_1)n$ , so  $G$  has linear stretch.  $G$  can also be computed by linear-size circuits because  $h_0$ ,  $h_1$ , and  $f$  can all be computed by linear-size circuits.

We note that it may be the case that one can only generate “good” one-way functions that satisfy Assumption 11 with constant probability; for instance, randomly selected bipartite graphs may only yield “good” OWFs with constant probability. One can still construct a family of PRGs from such a family of one-way functions, but the resulting PRGs will not be optimal because they will only be  $(2^{\Omega(\sqrt{n})}, 2^{-\Omega(\sqrt{n})})$  PRGs; see the full version for further details.

**PRGs from Regular One-Way Functions.** We have so far presented a construction of a PRG for exponentially hard one-way functions with certain preimage constraints. Using the hardcore and hash families outlined here that

can be computed by linear sized circuits, we can also modify a result of [19,16] to obtain asymptotically optimal PRGs for regular one-way functions with possibly unknown preimage size (recall that a one-way function  $f$  is *regular* if every every output  $f(x)$  has the same number of preimages. We refer the reader to the full for details<sup>2</sup>.

**Corollary 15.** *If  $f : \{0,1\}^n \rightarrow \{0,1\}^n$  is a  $\beta$ -exponential regular one-way function (with possibly unknown preimage size), then there exists a  $(2^{\Omega(n)}, 2^{-\Omega(n)})$  pseudorandom generator  $G$  with linear stretch that can be computed by linear-size circuits with  $O(1)$  oracle calls to  $f$ .*

**Acknowledgements.** The authors wish to thank Benny Applebaum, Andrej Bogdanov, Iftach Haitner, and Salil Vadhan for helpful discussions. This work was done in part while the first and second authors were at UCLA. The work of the first and third authors is supported in part by NSF grants CCF-0916574, IIS-1065276, CCF-1016540, CNS-1118126, CNS-1136174, and by US-Israel BSF grant 2008411. It was also supported by the OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Foundation Award, Teradata Research Award and Lockheed-Martin Corporation Research Award. The material contained herein is also based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0392. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. The work of the second author is supported by the European Research Council as part of the ERC project CaC (grant 259426), ISF grant 1361/10, and BSF grant 2008411.

## References

1. Alekhnovich, M.: More on average case vs approximation complexity. In: Proc. FOCS 2003, pp. 298–307 (2003)
2. Alekhnovich, M., Hirsch, E.A., Itsykson, D.: Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. J. Autom. Reasoning. 35(1-3), 51–72 (2005)
3. Applebaum, B.: Pseudorandom Generators with Long Stretch and Low Locality from Random Local One-Way Functions. In: Proc. STOC 2012, pp. 805–816 (2012)
4. Applebaum, B., Bogdanov, A., Rosen, A.: A Dichotomy for Local Small-Bias Generators. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 600–617. Springer, Heidelberg (2012)
5. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography in  $\text{NC}^0$ . SIAM J. on Computing 36(4), 845–888 (2006)
6. Applebaum, B., Ishai, Y., Kushilevitz, E.: On Pseudorandom Generators with Linear Stretch in  $\text{NC}^0$ . J. Comp. Compl. 17(1), 38–69 (2008)
7. Blum, M., Micali, S.: How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. SIAM J. on Computing 13(4), 850–864 (1985)

---

<sup>2</sup> We note that we can similarly modify a result of HILL for OWFs with *known* preimage size to yield asymptotically optimal PRGs as well; see the full version for details.

8. Bogdanov, A., Qiao, Y.: On the Security of Goldreich's One-Way Function. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) APPROX and RANDOM 2009. LNCS, vol. 5687, pp. 392–405. Springer, Heidelberg (2009)
9. Cook, J., Etesami, O., Miller, R., Trevisan, L.: Goldreich's One-Way Function Candidate and Myopic Backtracking Algorithms. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 521–538. Springer, Heidelberg (2009)
10. Cook, J., Etesami, O., Miller, R., Trevisan, L.: On the One-Way Function Candidate Proposed by Goldreich. ECCC, Report No. 175 (2012)
11. Cryan, M., Miltersen, P.B.: On Pseudorandom Generators in  $\text{NC}^0$ . In: Sgall, J., Pultr, A., Kolman, P. (eds.) MFCS 2001. LNCS, vol. 2136, pp. 272–284. Springer, Heidelberg (2001)
12. Goldreich, O.: Candidate One-Way Functions Based on Expander Graphs. ECCC, Report No. 90 (2000)
13. Goldreich, O.: Foundations of Cryptography. Cambridge U. Press, Cambridge (2001)
14. Goldreich, O., Krawczyk, H., Luby, M.: On the Existence of Pseudorandom Generators. SIAM J. on Computing 22(6), 1163–1175 (1993)
15. Goldreich, O., Levin, L.A.: Hard-core Predicates for any One-Way Function. In: Proc. STOC 1989, pp. 25–32 (1989)
16. Haitner, I.: New Implications and Improved Efficiency of Constructions Based on One-way Functions. Ph.D. Thesis (March 2008)
17. Haitner, I., Harnik, D., Reingold, O.: Efficient Pseudorandom Generators from Exponentially Hard One-Way Functions. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 228–239. Springer, Heidelberg (2006)
18. Haitner, I., Harnik, D., Reingold, O.: On the Power of the Randomized Iterate. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 22–40. Springer, Heidelberg (2006)
19. Haitner, I., Harnik, D., Reingold, O.: On the Power of the Randomized Iterate. SIAM J. on Computing 40(6), 1486–1528 (2011)
20. Haitner, I., Reingold, O., Vadhan, S.: Efficiency Improvements in Constructing Pseudorandom Generators from One-way Functions. In: Proc. STOC 2010, pp. 437–446 (2010)
21. Hastad, J., Impagliazzo, R., Levin, L.A., Luby, M.: A Psedorandom Generator From Any One-Way Function. SIAM J. on Computing 28(4), 1364–1396 (1999)
22. Holenstein, T.: Pseudorandom Generators from One-Way Functions: A Simple Construction for Any Hardness. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 443–461. Springer, Heidelberg (2006)
23. Holenstein, T., Maurer, U., Sjödin, J.: Complete Classification of Bilinear Hard-Core Functions. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 73–91. Springer, Heidelberg (2004)
24. Impagliazzo, R., Levin, L.A., Luby, M.: Pseudo-Random Generation From One-Way Functions (Extended Abstract). In: Proc. STOC 1989, pp. 12–24 (1989)
25. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Cryptography with Constant Computational Overhead. In: Proc. STOC 2008, pp. 433–442 (2008)
26. Mossel, E., Shpilka, A., Trevisan, L.: On epsilon-biased generators in  $\text{NC}^0$ . Random Struct. Algorithms 2(1), 56–81 (2006)
27. Näslund, M.: Universal Hash Functions & Hard Core Bits. In: Guillou, L.C., Quisquater, J.-J. (eds.) EUROCRYPT 1995. LNCS, vol. 921, pp. 356–366. Springer, Heidelberg (1995)

28. Panjwani, S.K.: An experimental evaluation of goldreich's one-way function. Technical report, IIT, Bombay (2001)
29. Vadhan, S., Zheng, C.J.: Characterizing Pseudoentropy and Simplifying Pseudorandom Generator Constructions. In: Proc. STOC 2012, pp. 817–836 (2012)
30. Yao, A.C.: Theory and application of trapdoor functions. In: Proc. FOCS 1982, pp. 80–91 (1982)

# A Fast Algorithm Finding the Shortest Reset Words

Andrzej Kisielewicz<sup>1,2,\*</sup>, Jakub Kowalski<sup>1</sup>, and Marek Szykuła<sup>1</sup>

<sup>1</sup> Department of Mathematics and Computer Science, University of Wrocław, Poland

<sup>2</sup> Institute of Mathematics and Computer Science, University of Opole, Poland

[andrzej.kisielewicz@math.uni.wroc.pl](mailto:andrzej.kisielewicz@math.uni.wroc.pl),  
[{kot,msz}@ii.uni.wroc.pl](mailto:{kot,msz}@ii.uni.wroc.pl)

**Abstract.** In this paper we present a new fast algorithm for finding minimal reset words for finite synchronizing automata, which is a problem appearing in many practical applications. The problem is known to be computationally hard, so our algorithm is exponential in the worst case, but it is faster than the algorithms used so far and it performs well on average. The main idea is to use a bidirectional BFS and radix (Patricia) tries to store and compare subsets. Also a number of heuristics are applied. We give both theoretical and practical arguments showing that the effective branching factor is considerably reduced. As a practical test we perform an experimental study of the length of the shortest reset word for random automata with  $n \leq 300$  states and 2 input letters. In particular, we obtain a new estimation of the expected length of the shortest reset word  $\approx 2.5\sqrt{n - 5}$ .

**Keywords:** Synchronizing automaton, synchronizing word, Černý conjecture.

## 1 Introduction

We deal with (complete deterministic) finite automata  $A = \langle Q, \Sigma, \delta \rangle$  with the state set  $Q$ , the input alphabet  $\Sigma$ , and the transition function  $\delta : Q \times \Sigma \rightarrow Q$ . The action of  $\Sigma$  on  $Q$  given by  $\delta$  is denoted simply by concatenation:  $\delta(q, a) = qa$ . This action extends naturally to the action  $qw$  of words for any  $w \in \Sigma^*$ . If  $|Qw| = 1$ , that is, the image of  $Q$  by  $w$  consists of a single state, then  $w$  is called a *reset* (or *synchronizing*) word for  $A$ , and  $A$  itself is called *synchronizing*. (In other words,  $w$  resets (synchronizes)  $A$  in the sense that, under the action of  $w$ , all the states are sent into the same state). The synchronizing property is very important, because it makes the automaton resistant to errors that could occur in an input word. After detecting an error a synchronizing word can be used to reset the automaton to its initial state. Synchronizing automata have many practical applications. They are used in robotics (for designing so-called part orienters) [2], bioinformatics (the reset problem) [3], network theory [11], theory of codes [10] etc.

---

\* Supported in part by Polish MNiSzW grant N N201 543038.

Theoretical research in the area is mainly motivated by the Černý conjecture stating that every synchronizing automaton  $A$  with  $n$  states has a reset word of length  $\leq (n - 1)^2$ . This conjecture was formulated by Černý in 1964 [4], and is considered the most longstanding open problem in the combinatorial theory of finite automata. So far, the conjecture has been proved only for a few special classes of automata and a general cubic upper bound  $(n^3 - n)/6$  has been established (see Volkov [24] for an excellent survey of the results, and Trahtman [23] for a recently found new cubic bound). Using computers the conjecture has been verified for small automata with 2 letters and  $n \leq 10$  states (and with  $k \leq 4$  letters and  $n \leq 7$  states [22]; see also [1] for  $n = 9$  states). It is known that, in general, the problem is computationally hard, since it involves an NP-hard decision problem. Recently, it has been shown that the problem of finding the length of the shortest reset word is  $\text{FP}^{\text{NP}[\log]}-\text{complete}$ , and the related decision problem is both NP- and coNP-hard [14].

On the other hand, there are several theoretical and experimental results showing that most synchronizing automata have relatively short reset words and those slowly synchronizing (with the shortest reset words of quadratic length) are rather exceptional [1]. An old result by Higgins [9] on products in transformation semigroups shows that a random automaton with an alphabet of size larger than  $2n$  has, with high probability, a reset word of length  $\leq 2n$ . More recently, it was proved that, for every  $\epsilon > 0$ , a random automaton with  $n$  states over an alphabet of size  $n^{0.5+\epsilon}$ , with high probability, is synchronizing and satisfies the Černý conjecture [20]. In computing reset words, either exponential algorithms finding the shortest reset words [19,22,12] or polynomial heuristics finding relatively short reset words [8,12,16,17,22] are widely used. The standard approach is to construct the power automaton and to compute the shortest path from the whole set state to a singleton [18,22,12,24]. Most naturally, the breadth-first-search method is used which starts from the set of all states of the given automaton and forms images applying letter transformations until a singleton is reached. Based on these ideas computation packages have been created (TESTAS [21] and recently developed COMPAS [5]). In [17], Roman uses a genetic algorithm to find a reset word of randomly generated automata and thus obtains upper bounds on the length of the shortest reset word.

A new interesting approach for finding the exact length using a SAT-solver has been applied recently by Skvortsov and Tipikin [19]. The problem of determining if an automaton has a reset word of length at most  $l$  is reduced to the SAT problem and the binary search for the exact length is performed. Using this approach, the following experimental study is done. For chosen numbers  $n$  of states from the interval  $[1, 100]$  random automata with 2 input letters are generated, checked if they are synchronizing, and if so, the shortest reset word is computed. The results directly contradict the conjecture made by Roman [17] that the mean length of the shortest reset word for a random  $n$ -state synchronizing automaton is linear and almost equal to  $0.486n$ . Skvortsov and Tipikin argue that their experiment based on a larger set of data shows that this length is actually sublinear and  $\approx 1.95n^{0.55}$ .

In this paper we present a new algorithm based on a bidirectional breadth-first-search. Implementing this idea requires efficiently solving the problem of storing and comparing resulted subsets of states. To this aim radix tries (also known as Patricia tries [13]) are used. We analyze the algorithm from both theoretical and practical sides. As the first test of efficiency we have performed experiments analogous to those done by Skvortsov and Tipikin. Due to the well performance of the algorithm we were able to generate and check one million automata for each  $n \leq 100$ , (compared with 200–2000 generated by Skvortsov and Tipikin), and we were able to test much larger automata with up to  $n = 320$  states. Our data confirm the hypothesis that the expected length of the shortest reset word is sublinear, but show that more precise is a smaller approximation  $\approx 2.5\sqrt{n - 5}$ . In addition, the larger set of data enables us to estimate the error and to show that for our approximation with high probability the error is very small. We also verify and discuss other results and claims of [19].

Our algorithm makes also possible to find a reset word of the shortest length (not only the length). Curiously, it works in polynomial time for known slowly synchronizing automata series [1]. So far, most of the empirical research in the area concerns automata with 2 input letters. Some researches suggest that automata with more letters may exhibit a different behavior. We plan to use the algorithm to perform an extensive research on automata with  $k > 2$  letters.

## 2 Algorithm

The algorithm gets an automaton  $A = \langle Q, \Sigma, \delta \rangle$  with  $n$  states and  $k$  input letters. First,  $A$  is checked if it is synchronizing using the well known (and efficient) algorithm [7]. If so, then we proceed to search for a synchronizing word of the shortest length. Here, one may perform the breadth-first search (BFS) on the power automaton of  $A$  starting from the set  $Q$  of all the states and computing successive images by the letters of the alphabet  $\Sigma$  (and recording the sequences of the letters applied). One may also search in the inverse (backward) direction starting from the singleton sets and computing successive preimages (this search will be referred to as IBFS). Both the searches have branching factor  $k$  (the number of input letters) and need to compute  $O(k^l)$  sets (or  $O(nk^l)$  in IBFS) to find a synchronizing word of the shortest length  $l$ . The idea behind bidirectional search is to perform two searches simultaneously and check if they meet. Then a synchronizing word may be found in only  $O(nk^{l/2})$  steps. However, to implement this idea there must be an efficient way to check each new subset to see if it already appears in the search tree of the other half of the search.

### 2.1 General Ideas

For each search we maintain the current list of subsets that can be obtained from the start in a given number of steps. Since the lists have a tendency to grow exponentially and to contain subsets obtained on earlier steps, it is more efficient to maintain additional lists of visited subsets (for each search) and to

use them to remove from the current lists redundant subsets. We have checked experimentally that it is a good strategy to decrease the branching factor.

To check if the two searches meet one needs to perform *subset checking*: after each step, BFS or IBFS, we check if a set on the current IBFS list *contains* a set on the current BFS list. If so, it means that there are words  $u, w \in \Sigma^*$  such that the image  $Qu$  is a subset of the preimage  $\{q\}w^{-1}$  for some  $q \in Q$ . Consequently,  $Quw = \{q\}$ , as required.

Since, in the bidirectional approach, subset checking must be performed anyway, it may be also applied to reduce lists using the following simple observation. If  $S$  and  $T$  are subsets of  $Q$  such that  $S \subseteq T$ , then  $|Tw| = 1$  implies  $|Sw| = 1$  for any  $w \in \Sigma^*$ . It follows that, for example, a subset on the IBFS list contains a subset on the BFS list if and only if – with respect to inclusion – a maximal element on the IBFS list contains a minimal element on the BFS list. Consequently, the only subsets on the BFS lists we need to consider are those minimal with respect to inclusion and the only subsets on the IBFS lists we need to consider are those maximal with respect to inclusion.

To store and check subsets on the lists we apply an efficient data structure known as *radix trie* (Patricia trie) [13]. We show that the *subset checking* operation (checking whether a given set  $S$  has a subset stored in the trie) and the dual *superset checking* (checking whether a given set  $S$  has a superset stored in the trie) are efficient enough for these structures to make a combination of the ideas presented above work well in practice.

This approach is fast but memory consuming. In order to also make the algorithm work efficiently for larger automata, when the memory limit is reached, the bidirectional approach is replaced by a sort of an inverse DFS search not involving the tries of visited subsets anymore. We also apply several technical optimizations and heuristics which yields a considerable speed-up. They are described in Section 3.

## 2.2 Radix Tries

A *radix trie* is a binary tree of the maximal depth  $n$  which stores subsets of a given  $n$ -set  $Q$  in its leaves. Having a fixed linear order of elements  $q_1, \dots, q_n \in Q$ , each subset  $S$  of  $Q$  encodes a path from the root to a leaf in the natural way: after  $i$  steps the path goes to the right child whenever  $q_i \in S$ , and goes to the left, otherwise. A radix trie is *compressed* in the sense that instead in a node at depth  $n$  it stores a subset in the first node that determines uniquely the subset in the stored collection (no other subset shares the same path as a prefix of the encoding); c.f. [13].

The insert operation for radix tries is natural and can be performed in at most  $n$  steps. The *subset checking* operation is performed by a depth-first-search checking if the given set  $S \subseteq Q$  contains a subset stored in the visited leaf. An essential advantage is that the search does not need to branch into the right child of a node if the checked subset  $S$  does not contain the state corresponding to the current level. The *superset checking* operation (for IBFS) is done in the dual way. These issues are discussed in more detail in 2.3.

**Algorithm 1.** The main part

---

**Input**  $A = \langle Q, \Sigma, \delta \rangle$  – a synchronizing automaton with  $n = |Q|$  states and  $k = |\Sigma|$  input letters.  
**Input**  $\text{ maxlen}$  – maximum length of words to be checked.

▷ Initialize four radix tries to store and handle subsets of  $Q$ :

- 1:  $T_c \leftarrow \text{EMPTYTRIE}$  ▷ BFS current trie
- 2:  $T_v \leftarrow \text{EMPTYTRIE}$  ▷ BFS visited trie
- 3:  $T_{ic} \leftarrow \text{EMPTYTRIE}$  ▷ IBFS current trie
- 4:  $T_{iv} \leftarrow \text{EMPTYTRIE}$  ▷ IBFS visited trie
- 5:  $T_c.\text{INSERT}(Q)$
- 6:  $T_v.\text{INSERT}(Q)$
- 7: **for all**  $q \in Q$  **do**
- 8:      $T_{ic}.\text{INSERT}(\{q\})$
- 9:      $T_{iv}.\text{INSERT}(\{q\})$
- 10: **end for**
- 11: **for**  $l \leftarrow 1$  **to**  $\text{ maxlen}$  **do**
- 12:     **if** estimated time of the BFS step is smaller than that of IBFS **then**
- 13:         BFS\\_STEP( $T_c, T_v$ ) ▷ Modify BFS tries; minimize  $T_c$  using  $T_v$
- 14:     **else**
- 15:         IBFS\\_STEP( $T_{ic}, T_{iv}$ ) ▷ Modify IBFS tries; minimize  $T_{ic}$  using  $T_{iv}$
- 16:     **end if**
- 17:     **for all**  $S \in T_{ic}$  **do** ▷ The goal test loop
- 18:         **if**  $T_c.\text{CONTAINS\_SUBSET\_OF}(S)$  **then**
- 19:             **return**  $l$  ▷ The length of the shortest reset word
- 20:         **end if**
- 21:     **end for**
- 22: **end for**
- 23: **return** "No synchronizing word of length  $\leq \text{ maxlen}$ "

---

### 2.3 Description

The main part of the algorithm is given in Algorithm 1. To make it clearer we restrict the task to finding the *shortest length* of a reset word only. Yet, the algorithm can be easily modified to return also a reset word of such length (see 2.4).

We use, in principle, four radix tries  $T_c, T_v, T_{ic}, T_{iv}$  to maintain the BFS current, BFS visited, IBFS current, and IBFS visited lists, respectively. After initializing the tries we enter a loop consisting of at most  $\text{ maxlen}$  steps (line 11). In each step we perform a step of the BFS procedure or IBFS procedure depending on comparison of estimated expected execution time of both steps, which we discuss in 3.1.

With no regard if BFS or IBFS step was performed recently, in lines 17–21 of Algorithm 1, the same goal test loop is performed. For each  $S$  in  $T_{ic}$ , the procedure  $T_c.\text{CONTAINS\_SUBSET\_OF}(S)$  is executed, which checks if  $T_c$  contains a subset of  $S$ . If so, we claim that  $l$  is the shortest length of a rest word for  $A$ . To prove this we need to analyze the content of the BFS and IBFS steps.

In BFS step (Algorithm 2), for each set  $S'$  in the current BFS trie and for each input letter  $a$  we compute the image  $S = S'a$  and insert it to the list  $L$ . For each set  $S \in L$  we check if a subset of  $S$  is already in the BFS visited trie. If so, we skip it. If not, we insert  $S$  into the BFS visited trie and in the (newly formed; line 9) BFS current trie  $T_c$ . Processing elements of  $L$  (line 10) in *ascending cardinality order* is a heuristic aimed in getting more subsets skipped in the checking subset procedure in line 11, and in consequence, to deal with smaller structures. It also guarantees that  $T_c$  contains only minimal sets in terms of inclusion (the proof of this fact and all other proofs will be given in the extended version of this paper).

After executing lines 10-15 of Algorithm 2 the trie  $T_v$  may contain some redundant subsets (which are not minimal with respect to inclusion). Therefore in lines 16-18 we have an additional procedure to reduce  $T_v$  completely.

The procedure  $T_v.\text{REDUCE}$  consists of two steps. First, we form a list of elements of  $T_v$  using a DFS-search from the left to the right (smaller subsets first). This guarantees that if  $S$  precedes  $T$  on the list then  $S$  does not contain  $T$ . Hence the only pairs of comparable elements on the list are those with  $S$  preceding  $T$  and  $S \subset T$ . In the second step we insert the elements from the list into the empty  $T_v$  depending on the result of subset checking performed before each insertion. This guarantees that if a subset  $S$  of  $T$  is inserted then  $T$  will be skipped on the later step. Hence the resulting trie  $T_v$  contains no comparable subsets, as required.

Unfortunately, this procedure applied for such a large trie as  $T_v$  (which may be of exponential size in terms of  $n$ ) may be time-consuming. We found experimentally that if the trie has not grown too large since the last reduction it is more effective to process a larger trie rather than to perform reduction. In our implementation we perform it after the first step and then only when  $T_v$  contains at least  $k$  times more sets since it had after the last reduction (which is the worse case for one step with branching factor  $k = |\Sigma|$ ).

The IBFS step is dual and completely analogous. In line 10 ascending cardinality order is replaced by descending one, in line 5 we compute preimages instead of images, and in line 11 subset checking is replaced by superset checking.

One can prove the following

**Theorem 1.** *Given a synchronizing  $n$ -state automaton  $A = \langle Q, \Sigma, \delta \rangle$ , Algorithm 1 returns the shortest length of a reset word for  $A$  or reports that no such a word of length  $\leq \text{maxlen}$  exists.*

## 2.4 Finding a Reset Word

In order to find a reset word of the found minimal length  $l$ , one needs to apply the following slight modification to the algorithm described above. The main point is that together with the sets stored in the current tries we need to store also the words assigned to these sets. To this end, in line 5 of Algorithm 2 (and analogously in the IBFS procedure) we assign to  $S'$  the word obtained by concatenating the word assigned earlier to  $S$  with the letter  $a$  (at the end or at the beginning, respectively). When the goal is reached, the two words are

**Algorithm 2.** BFS step procedure

---

```

1: procedure BFS_STEP( $T_c, T_v$ )
2:    $L \leftarrow \text{EMPTYLIST}$                                  $\triangleright$  The list of all new images
3:   for all  $S' \in T_c$  do
4:     for all  $a \in \Sigma$  do
5:        $S \leftarrow \delta(S', a)$                            $\triangleright$  Compute the image of  $S'$  by the letter  $a$ 
6:        $L.\text{INSERT}(S)$ 
7:     end for
8:   end for
9:    $T_c \leftarrow \text{EMPTYTRIE}$ 
10:  for all  $S \in L$  in ascending cardinality order do
11:    if not  $T_v.\text{CONTAINS\_SUBSET\_OF}(S)$  then
12:       $T_v.\text{INSERT}(S)$ 
13:       $T_c.\text{INSERT}(S)$ 
14:    end if
15:  end for
16:  if  $T_v$  has grown large since the last reduction then
17:     $T_v.\text{REDUCE}$ 
18:  end if
19: end procedure

```

---

simply merged to form the required reset word. Of course, instead of complete words, with each set we store only a letter and a pointer to the previous part of the word. From these the word is reconstructed when we reach the goal. We note that in this way the asymptotic time and space complexity of the algorithm remain the same.

### 3 Heuristics and Optimizations

In addition to the main part of the algorithm described in the previous section we use a number of heuristics and optimizations. They are justified both by experiments and theoretical arguments. Altogether they can reduce computation time by a factor of at least 25 relative to the implementation without these optimizations. We describe briefly only the most important of them.

#### 3.1 Estimation of Expected Step Time

To decide which step will be performed in line 12 of the Algorithm 1 we follow the greedy strategy choosing this step whose execution time, together with the goal test, seems to be smaller at the moment. We use a rough estimation of expected execution time by calculating upper bounds for the expected number of visited nodes in subset checking operations, under some simplifying assumptions. Since all other operations in the steps in question are linear in terms of  $n$  and the sizes of the current lists, subset checking are the most time consuming operations. The base for the estimation is the following theoretical result we have established. (A

set  $S \subset X$  is a random subset of  $X$  with Bernoulli distributions in  $[q, r]$  if each element  $x$  of  $X$  is a member of  $S$  with probability  $p_x \in [q, r]$ .)

**Theorem 2.** Let  $p, q, r \in (0, 1)$  be such that  $q \leq r$  and  $q > pr$ . Let  $\mathcal{F}$  be a family of  $m$  random subsets of a given set  $X$  with Bernoulli distributions in  $[q, r]$ , and let  $S$  be a random subset of  $X$  with Bernoulli distributions in  $[0, p]$ . Then in the trie constructed for the family  $\mathcal{F}$ , the expected number of visited nodes by the subset checking procedure for  $S$  is at most

$$\left( \frac{1+p}{p} + \frac{1}{q-pr} \right) m^{\log_w (1+p)},$$

where  $w = \frac{1+p}{1+pr-q}$ .

In our empirical observations this optimization reduces computation time by an average of 70% relative to the implementation performing the BFS and IBFS steps alternately. It usually leads to perform slightly more BFS steps, since average sizes of subsets decrease much faster in BFS than increase in IBFS. By a result of Higgins after applying two BFS steps the average size of subsets not greater than  $0.55n$  (see [9]). Our empirical observations show that the two searches meet when the sizes of subsets are as small as  $0.03n$ . This fact is also the reason why in the goal test we decided to use subset checking of  $T_c$  rather than superset checking of  $T_{ic}$  (subset checking does not require branching in subtrees corresponding to elements not belonging to the queried set).

### 3.2 Adding the IDFS Phase

This is the most important optimization improving not only the performance, but also modifying the general idea. Bidirectional BFS works if we have no limit on memory resources. Since the number of sets stored in the tries grows exponentially with the number of steps performed, for large automata, we can easily run out of memory. To deal with this, we change the search strategy when we reach the memory limit. Rather than to continue BFS searches we switch to depth-first search, which has restricted memory requirements, and may use the subsets and words computed so far. Moreover, assuming the Černý conjecture, we may impose an initial limit on the depth of the search, which allows to make the DFS search *complete*. After each recursive call, when a shorter reset word is found, the limit on the depth of the search is suitably decreased. The search is finished when no limit decreasing is possible and all paths of the limited DFS are exhausted. The search returns either *the shortest reset word* or a counterexample to the Černý conjecture. The IDFS phase is used also to reduce the computation time of the algorithm (even if we are far from reaching the memory limit). This will be discussed in subsection 3.5.

Our experiments show that it is more efficient to apply the *inverse* DFS, that is, one starting from the sets in  $T_{ic}$  and computing the preimages to find a set containing a member of  $T_c$  (rather than the *forward* DFS starting from the sets in  $T_c$  and computing images to find a set contained in a member of  $T_{ic}$ ).

An important modification is that we perform search on partial lists of subsets making use of all available memory rather than on single subsets. This gives an additional boost.

### 3.3 Reduction of the Automaton

If the input automaton is not strongly connected, after some steps of BFS it can be reduced to a smaller automaton without the states not involved in computation anymore. More precisely, we can remove the states which are not reachable from any state in any subset in the current BFS list. So, at the beginning, before the main loop of Algorithm 1 (line 11), we perform a few steps of BFS and when the size of  $T_c$  is larger than  $sn$ , where  $s$  is an experimentally established constant, we check if there are unreachable states in  $Q$ . This is done by the standard DFS search on  $Q$ . If this is the case, we create a reduced automaton  $A'$  removing the unreachable states, and rebuild all the tries to make them compatible with the reduced automaton. Then, the algorithm may continue using the parameters computed so far.

Our experiments show that after the first reduction the automaton is usually strongly connected (and no further reduction of this kind can be done). Yet, this optimization is efficient since we have proved that the fraction of strongly connected automata to all automata with  $n$  states tends to 0 as  $n$  goes to infinity, and that the size of the minimal strongly connected component is on average less than  $1 - 1/e^k$  (provided most automata are synchronizing). From our experiments it follows that for synchronizing automata with  $k = 2$  this size is  $\approx 0.7987n$ . Thus, for example, automata with  $n = 200$  states are reduced on average by as much as 40 states.

### 3.4 Reordering of the States

Efficiency of operations on radix tries depends on the order in which the input automaton's states are processed. We found that the subset checking is performed faster if the states occurring more frequently in queried subsets are later in the ordering. This is because radix tries tends to have logarithmic height (cf. [6]), and the states at the end in the ordering are rarely or never checked. As a result, the "effective size" of the queried sets is smaller. To establish frequencies of occurrences of states, and a preferred initial order based on them, we use a stationary distribution of a Markov chain based on the underlying digraph of the automaton. The details will be given in the extended version of the paper. This optimization is performed before the bidirectional search phase.

The situation changes completely during the IDFS phase, when the trie  $T_c$  is fixed and does not change anymore. The frequencies of occurrences of the subsets in  $T_c$  may be computed exactly. This leads to a different reordering. Both reorderings have been confirmed as optimal by experiments. They show that these optimizations reduce computation time by an average of 27%.

### 3.5 Using Heuristic Algorithms and IDFS Shortcut

In order to save a step of search computation we may use known heuristic algorithms to find quickly a good bound for search depth. Therefore, at the beginning of the algorithm, before starting the bidirectional search, we apply a few polynomial time algorithms finding upper bounds for the length of the shortest reset word. In our implementation we use Eppstein algorithm [7], FastSynchro algorithm [12] and our procedure Cut-Off IBFS. The latter is the standard IBFS search with cutting the branches of the search with smallest subsets. This may spare one step in bidirectional search, if the heuristic algorithms find the shortest word.

Yet, more importantly, combined with the IDFS phase, this makes possible to reduce the computation time by several orders of magnitude. Knowing that bidirectional search is close to end it is profitable to switch to IDFS phase: at the end the IDFS works much faster, since we do not need to check visited sets and do not need to reconstruct  $T_c$  anymore. We call this optimization the *shortcut*. Between steps we use an estimate if it is faster to continue the bidirectional phase or to switch to IDFS phase. Note that the IDFS has a lower constant factor, but the branching factor is equal to  $k$ . So, it slows the search if started too early. For estimation we use the formula in Theorem 2. Our experiments show that this optimization reduces computation time by as much as 89%.

## 4 Complexity

The efficiency gain of the algorithm relies mainly on two properties of the majority of automata. First, the average size of subsets decreases fast during the first BFS steps, but increases slow during IBFS steps (cf. subsection 3.1). Due to this fact the maintained subsets are usually small. Second, the branching factors of both BFS and IBFS are less than  $k$ , because of skipping redundant visited sets. Both of the properties are hard to study in a theoretical way, we however have observed them in series of experiments.

To provide a theoretical argument we analyze here the expected running time of the algorithm under some artificial assumptions. We give an upper bound for the bidirectional search only, which is a rough estimate of the expected time, but shows a significant impact of the automata properties on performance. The following assumptions are made:

1. The input is a synchronizing automaton with  $n$  states on  $k$  letters.
2. The overall branching factor is  $r$  in each step of both BFS and IBFS,  $1 < r < k$ . This corresponds to an effective branching factor, which in view of our experiments is considerably less than  $k$ .
3. The sets in the tries  $T_c, T_v$  and  $T_{ic}, T_{iv}$  have random Bernoulli distribution: in each step, they contain any given state with probability  $0 < p_c < 1$  (for BFS steps) and  $0 < p_{ic} < 1$  (for IBFS steps). We assume also that  $p_{ic} \leq p_c$ .
4. The steps of BFS and IBFS are performed alternatingly, starting from BFS.
5. No reductions of the visited tries are made and no IDFS phase is performed.

While the assumptions 2-3 are purely theoretical, they may be treated as an idealization of a typical situation. Using these assumption, denoting by  $l$  the length of the shortest reset word of the automaton, we can prove that there exists an integer  $0 < d < 1$ , depending on probabilities  $p_c, p_{ic}$ , such that the following holds.

**Theorem 3.** *Under the assumptions (1-5) above, and with  $l$  denoting the length of the shortest reset word of the automaton, the expected time complexity of the algorithm is  $O(kn^2r^{l(1+d)/2})$ , and the space complexity is  $O(n(k+n)+nr^{l/2})$ .*

We can observe that the expected time is exponential with regard to the length  $l$ , but the exponent is less than  $l$ , since  $(1+d)/2 < 1$ . It is an improvement over the standard BFS algorithm, which has time bound  $O(knR^l)$  (assuming we can check visited sets in constant time). Moreover the standard algorithm usually has a larger branching factor  $R > r$ , since strict supersets of visited sets are not skipped. The expected space complexity also yields an improvement in comparison to the  $O(nR^l)$  space bound for the standard BFS.

While, generally, our algorithm is exponential in the length  $l$  of the shortest reset word, surprisingly, it works fast in polynomial time for the known series of *slowly synchronizing automata*, that is those with  $l$  close to the Černý bound. These are automata  $\mathcal{C}_n$  (the Černý automaton),  $\mathcal{W}_n, \mathcal{D}'_n, \mathcal{D}''_n$ , and  $\mathcal{B}_n$  introduced in [1].

**Theorem 4.** *For the class of the Černý automata  $\mathcal{C}_n$ , and the classes a  $\mathcal{W}_n, \mathcal{D}'_n, \mathcal{D}''_n$ , and  $\mathcal{B}_n$  introduced in [1] the algorithm works in  $O(n^4)$  time and  $O(n^3)$  space.*

The proof is based on the exact description of the heuristic mentioned in 3.1, which shows that for each of the mentioned slowly synchronizing automata the algorithm performs mainly IBFS steps (rather than BFS), and the IBFS lists keep containing only one or two sets (due to reductions of visited subsets).

## 5 Experiments

We performed a series of the following experiments for various  $n \leq 320$ . For a given  $n$ , we generate a random automaton  $A$  with  $n$  states and 2 input letters, check whether  $A$  is synchronizing and if so, we find the minimal length of a reset word using the algorithm described in Section 2. On the basis of the obtained results we estimate the expected length of the shortest reset word.

### 5.1 Computations

In the experiments we have used the standard model of random automata, where for each state and each letter all the possible transitions are equiprobable. A random automaton with  $n$  states and 2 input letters can be then represented as a sequence of  $2n$  uniformly random natural numbers from  $[0, n - 1]$ . To generate high quality random sequences we have used the WELL number generator

[15] (variants 1024 and 19937) seeded by random bytes from `/dev/random` device. For comparison, recall that Skvortsov and Tipikin, in their experimental study [19], have generated and checked the following numbers of random automata: 2000 automata for each  $n \in \{1, 2, \dots, 20, 25, 30, \dots, 50\}$ , 500 automata for each  $n \in \{55, 60, 65, 70\}$ , and 200 automata for each  $n \in \{75, 80, \dots, 100\}$ . In our experiment, up to 7 states, we have computed exact results checking all automata. For each  $8 \leq n \leq 100$  we checked one million automata, and for each  $101 \leq n \leq 260$  and  $n = 265, 270, \dots, 320$  we checked 10000 automata. Our computations have been performed mostly on 16 computers with Intel(R) Core(TM) i7-2600 CPU 3.40GHz 4 cores and 16GB of RAM. The algorithm was implemented in C++ and compiled with g++. Distributed computations were managed by a dedicated server and clients applications written in Python.

The average computation time is about 100 or 1000 times faster than the time of Trahtman's program TESTAS [21,22] for automata with 50 states. The reduction to SAT used in [19] seemed to be the fastest recently known algorithm and the reported average time for 50 states automata is 2.7 seconds, and for 100 states automata is 70 seconds. Our comparable results are less than 0.006 and 0.07 seconds, respectively (we have used faster machines, but only about twice as fast). The Table 1 presents a rough comparison. The average times are relatively small because of rare occurrences of slowly synchronizing automata. We present also the maximum computation time.

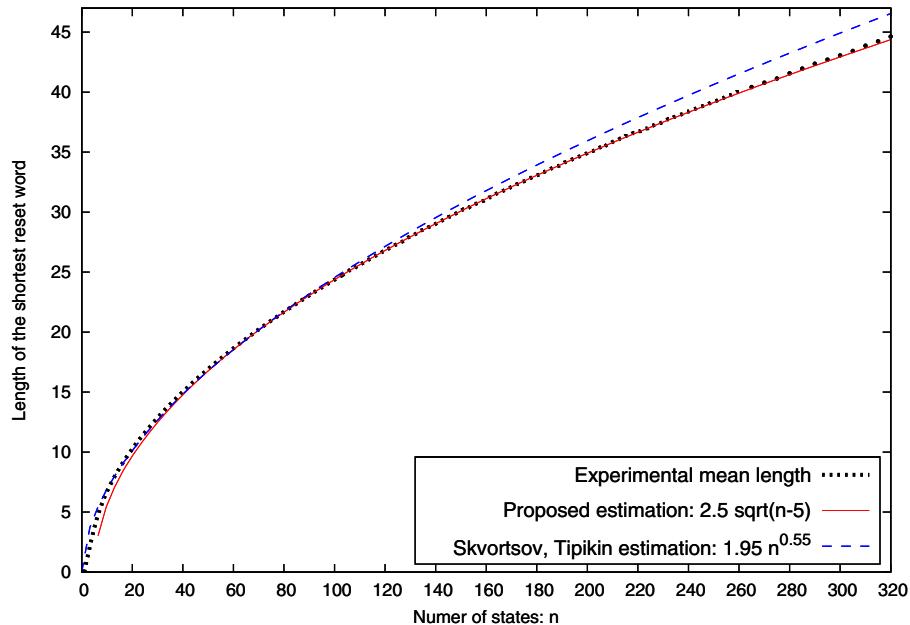
**Table 1.** Comparison of average and maximum computation time for random automata

$n$	50	100	150	200	250	300
TESTAS ([21])	1.4 s	time-out	—	—	—	—
SAT reduction ([19])	2.7 s	70 s	—	—	—	—
Our average time	0.005 s	0.06 s	0.469 s	2.88 s	31.637 s	596.249 s
Our maximum time	0.26 s	3.79 s	10.12 s	159.670 s	5 h 19 min	7 h 55 min

## 5.2 Results

Our experiment confirms that for the standard random automata model  $\mathbb{A}(n)$  on the binary alphabet the probability that the automaton is synchronizing seems to tend to 1 as the number  $n$  of states grows. This conjecture is posed in [19], but we have heard it earlier from Peter Cameron during BCC conference in Exeter 2011. For  $n = 100$ , 2250 of one million automata turned out to be non-synchronizing (0.225%), and for  $n = 300$ , only five of 10000 automata. The graphical representations of our experiments in this respect forms a smooth curve very fast converging to 1. We observe also that random automata mostly are not strongly connected.

The main result of our experiments is the estimation of the expected length of the shortest reset word. We deal with the infinite sequence of random variables  $\ell(n)$  defined as the length of the shortest reset word for a random synchronizing automaton with  $n$  states. We have observed that the approximation  $\mathbb{E}[\ell(n)] \approx 1.95n^{0.55}$  proposed in [19] is inflated. Based on currently available



**Fig. 1.** Experimental mean length of the shortest reset words compared with estimations

data, we propose a new more precise experimental approximation for the expected length  $\mathbb{E}[\ell(n)] \approx 2.5\sqrt{n-5}$ . A comparison of the estimations with the experimentally obtained mean length is given in Figure 1. We observe also that our result suggest that the expected length may belong to  $\Theta(\sqrt{n})$ .

In contrast with the experiments by Skvortsov and Tipikin [19], our experiments allow also to obtain a good estimation of the approximation error. Making use of the well-known Hoeffding's inequality, we obtain the following:

**Theorem 5.** *Let  $ML(n)$  denotes the mean length of the shortest reset word of the automata in the sample of  $m$  randomly generated synchronizing  $n$ -state automata. If the ratio of the automata with the length of the shortest reset word larger than  $M_n$  to all automata in the sample does not exceed  $r$ , then with probability at least  $1 - p$*

$$|ML(n) - \mathbb{E}[\ell(n)]| \leq M_n(1-r)\sqrt{\frac{\log(2/p)}{2m}} + \frac{n^3}{6}r.$$

Assuming the Černý conjecture in the last term  $n^3/6$  may be replaced by  $(n-1)^2$  (giving essentially better estimation). Let us take  $n = 100$ ,  $m = 10^6$  and  $p = 0.0001$ . Since, with probability  $q = (1-r)^m$  the ratio of the automata with the shortest reset word longer than  $M_n$  is less than  $r$ , one may see that for  $1/r \geq 100975$ ,  $q < 0.0001$ . Hence, with high probability  $1/r > 100975$ , and taking

into account the experimental value  $M_{100} = 41$ , the error is less than 1.75 (or 0.19 assuming the Černý conjecture). This means that with high probability the expected length of the shortest reset word for synchronizing automata with  $n = 100$  states is close to our experimental result  $ML(100) = 24.34$ . Comparing this with the results of Skvortsov and Tipikin [19], we note that, for automata with 100 states, they also have obtained the expected length close to 24, but the small size of their sample  $m = 200$  does not allow any reasonable estimation of the error. Other interesting claims of [19] concerning the variance and approximation of  $\ell(n)$  will be discussed in the extended version of the paper.

## References

1. Ananichev, D., Gusev, V., Volkov, M.: Slowly synchronizing automata and digraphs. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 55–65. Springer, Heidelberg (2010)
2. Ananichev, D., Volkov, M.: Synchronizing monotonic automata. In: Ésik, Z., Fülöp, Z. (eds.) DLT 2003. LNCS, vol. 2710, pp. 111–121. Springer, Heidelberg (2003)
3. Benenson, Y., Adar, R., Paz-Elizur, T., Livneh, Z., Shapiro, E.: DNA molecule provides a computing machine with both data and fuel. Proceedings of the National Academy of Sciences 100(5), 2191–2196 (2003)
4. Černý, J.: Poznámka k homogénnym eksperimentom s konečnými automatami. Matematicko-fyzikálny Časopis Slovenskej Akadémie Vied 14(3), 208–216 (1964) (in Slovak)
5. Chmiel, K., Roman, A.: COMPAS - A computing package for synchronization. In: Domaratzki, M., Salomaa, K. (eds.) CIAA 2010. LNCS, vol. 6482, pp. 79–86. Springer, Heidelberg (2011)
6. Devroye, L.: A note on the average depth of tries. Computing 28, 367–371 (1982)
7. Eppstein, D.: Reset sequences for monotonic automata. SIAM Journal on Computing 19, 500–510 (1990)
8. Gerbush, M., Heeringa, B.: Approximating minimum reset sequences. In: Domaratzki, M., Salomaa, K. (eds.) CIAA 2010. LNCS, vol. 6482, pp. 154–162. Springer, Heidelberg (2011)
9. Higgins, P.: The range order of a product of i-transformations from a finite full transformation semigroup. Semigroup Forum 37, 31–36 (1988)
10. Jürgensen, H.: Synchronization. Information and Computation 206(9-10), 1033–1044 (2008)
11. Kari, J.: Synchronization and stability of finite automata. Journal of Universal Computer Science 8(2), 270–277 (2002)
12. Kudlacik, R., Roman, A., Wagner, H.: Effective synchronizing algorithms. Expert Systems with Applications 39(14), 11746–11757 (2012)
13. Morrison, D.R.: PATRICIA – practical algorithm to retrieve information coded in alphanumeric. Journal of the ACM 15, 514–534 (1968)
14. Olszewski, J., Ummels, M.: The complexity of finding reset words in finite automata. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 568–579. Springer, Heidelberg (2010)
15. Panneton, F., L'Ecuyer, P., Matsumoto, M.: Improved long-period generators based on linear recurrences modulo 2. ACM Transactions on Mathematical Software 32(1), 1–16 (2006)

16. Roman, A.: New algorithms for finding short reset sequences in synchronizing automata. In: International Enformatika Conference (Prague), pp. 13–17 (2005)
17. Roman, A.: Genetic algorithm for synchronization. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 684–695. Springer, Heidelberg (2009)
18. Sandberg, S.: Homing and synchronizing sequences. In: Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A. (eds.) Model-Based Testing of Reactive Systems. LNCS, vol. 3472, pp. 5–33. Springer, Heidelberg (2005)
19. Skvortsov, E., Tipikin, E.: Experimental study of the shortest reset word of random automata. In: Bouchou-Markhoff, B., Caron, P., Champarnaud, J.-M., Maurel, D. (eds.) CIAA 2011. LNCS, vol. 6807, pp. 290–298. Springer, Heidelberg (2011)
20. Skvortsov, E., Zaks, Y.: Synchronizing random automata. Discrete Mathematics and Theoretical Computer Science 12(4), 95–108 (2010)
21. Trahtman, A.N.: A package TESTAS for checking some kinds of testability. In: Champarnaud, J.-M., Maurel, D. (eds.) CIAA 2002. LNCS, vol. 2608, pp. 228–232. Springer, Heidelberg (2003)
22. Trahtman, A.N.: An efficient algorithm finds noticeable trends and examples concerning the Černý conjecture. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 789–800. Springer, Heidelberg (2006)
23. Trahtman, A.N.: Modifying the upper bound on the length of minimal synchronizing word. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 173–180. Springer, Heidelberg (2011)
24. Volkov, M.V.: Synchronizing automata and the Černý conjecture. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 11–27. Springer, Heidelberg (2008)

# The Discrete Voronoi Game in a Simple Polygon\*

Aritra Banik<sup>1</sup>, Sandip Das<sup>1</sup>, Anil Maheshwari<sup>2</sup>, and Michiel Smid<sup>2</sup>

<sup>1</sup>Indian Statistical Institute, Kolkata 700108, India

{aritra.banik,sandip.das.69}@gmail.com

<sup>2</sup>School of Computer Science, Carleton University, Ottawa, ON K1S 5B6, Canada  
{anil,michiel}@scs.carleton.ca

**Abstract.** Let  $P$  be a simple polygon with  $m$  vertices and let  $\mathcal{U}$  be a set of  $n$  points in  $P$ . We consider the points of  $\mathcal{U}$  to be “users”. We consider a game with two players  $\mathcal{P}_1$  and  $\mathcal{P}_2$ . In this game,  $\mathcal{P}_1$  places a point facility inside  $P$ , after which  $\mathcal{P}_2$  places another point facility inside  $P$ . We say that a user  $u \in \mathcal{U}$  is served by its nearest facility, where distances are measured by the geodesic distance in  $P$ . The objective of each player is to maximize the number of users they serve. We show that for any given placement of a facility by  $\mathcal{P}_1$ , an optimal placement for  $\mathcal{P}_2$  can be computed in  $O(m + n(\log n + \log m))$  time. We also provide a polynomial-time algorithm for computing an optimal placement for  $\mathcal{P}_1$ .

## 1 Introduction

In a facility location problem, we are interested in finding a placement of a set of facilities so that, for a given set of users, certain optimality criteria are met. In a typical geometric facility location problem, the facilities and users are modeled as points. Each user is served by its *nearest* facility, with respect to an appropriate distance measure (e.g., Euclidean distance). Consequently, each facility has its *service zone*, consisting of the set of users that are served by it. The aim is to place the facilities so that certain optimality criteria are satisfied.

The *Voronoi game* is a competitive facility location problem introduced by Ahn et al.[1]. Given a user space, two players,  $\mathcal{P}_1$  and  $\mathcal{P}_2$ , sequentially place a set of point facilities. These facilities partition the user space into a set of regions, such that all users within a region are served by a particular facility. The objective of each player is to maximize the total service zone of all its facilities. This problem is generally intractable. Teramoto, Demaine, and Uehara [9] have shown that even if the underling user space is a graph, finding a winning strategy of  $\mathcal{P}_2$  (even for a very restricted case) is NP hard. Similar results can also be found in a seminal paper of Hakimi [7].

The discrete version of the Voronoi game is studied by Banik, Battacharya and Das [3]. Their user space is a line containing a set  $\mathcal{U}$  of  $n$  point users. Each of the players  $\mathcal{P}_1$  and  $\mathcal{P}_2$  can place  $k = O(1)$  point facilities. First,  $\mathcal{P}_1$  chooses a set

---

\* Research supported by NSERC and DFAIT Commonwealth Scholarship.

$F_1$  of  $k$  facilities, after which  $\mathcal{P}_2$  chooses a set  $F_2$  of  $k$  facilities, disjoint from  $F_1$ . The *payoff* of  $\mathcal{P}_2$  is defined to be the cardinality of the set of points in  $\mathcal{U}$  which are closer to some facility owned by  $\mathcal{P}_2$  than to every facility owned by  $\mathcal{P}_1$ . The payoff of  $\mathcal{P}_1$  is the number of users in  $\mathcal{U}$  minus the payoff of  $\mathcal{P}_2$ . The objective of both players is to maximize their respective payoffs. Banik et al. show that, if the sorted order of points in  $\mathcal{U}$  along the line is given, an optimal strategy of  $\mathcal{P}_2$  for any given placement of facilities of  $\mathcal{P}_1$  can be computed in linear time. They also provide results for determining an optimal strategy for  $\mathcal{P}_1$ .

Given a set of existing facilities, the problem of placing a set of new facilities, to maximize the number of users served by the new ones, has been actively studied. Cabello et al. [5] study the case when only one new facility by  $\mathcal{P}_2$  is introduced. This problem is referred to as the MaxCov problem. They have shown that the optimal placement for the new facility can be found in  $O(n^2)$  time. The 2-MaxCov problem, which considers the problem of placing two new facilities, has been studied by Bhattacharya and Nandy [4]. Recently Bandyapadhyay et al.[2] studied the one round discrete Voronoi game for graphs. Discrete Voronoi game for paths have been studied in the paper [8].

In this paper, we consider the Voronoi game, where the underlying user space is a simple polygon  $P$  with distance measure defined to be the geodesic (i.e., shortest-path) distance in  $P$ . The game consists of a set  $\mathcal{U}$  of  $n$  point-users inside  $P$ , and two players  $\mathcal{P}_1$  and  $\mathcal{P}_2$ . Initially,  $\mathcal{P}_1$  places a set  $F_1$  of  $k$  point-facilities, after which  $\mathcal{P}_2$  places a set  $F_2$  of  $k$  point-facilities, where  $F_1 \cap F_2 = \emptyset$ . Each user  $u \in \mathcal{U}$  is served by the nearest facility according to the nearest neighbor rule (i.e., by the facility which is at the least geodesic distance from  $u$ ).

**Definition 1.** (*Service zone*) For each facility  $f \in F_1 \cup F_2$ , we define its service zone  $\mathcal{U}_{F_1 \cup F_2}(\{f\})$  to be the set of users in  $\mathcal{U}$  that are closer to  $f$  than to any other facility of  $F_1 \cup F_2$ .

Given a set  $S \subseteq F_1 \cup F_2$ , we define the service zone of  $S$  to be the set of users which are assigned to one of the facilities in  $S$ , i.e.,  $\mathcal{U}_{F_1 \cup F_2}(S) = \bigcup_{f \in S} \mathcal{U}_{F_1 \cup F_2}(\{f\})$ .

Ties will be broken in favor of the facility placed later. With this definition, the problem considered in this paper can be formally described as follows.

**Definition 2.** One Round Discrete Voronoi Game for a Simple Polygon  $P$ : Given a set  $\mathcal{U}$  of  $n$  point-users and two players  $\mathcal{P}_1$  and  $\mathcal{P}_2$ , having  $k$  facilities each,  $\mathcal{P}_1$  chooses a set  $F_1$  of  $k$  point-facilities in  $P$ , after which  $\mathcal{P}_2$  chooses a set  $F_2$  of  $k$  point-facilities in  $P$ , where  $F_1 \cap F_2 = \emptyset$ .

- (a) Given any choice of  $F_1$  by  $\mathcal{P}_1$ , the objective of  $\mathcal{P}_2$  is to choose a set  $S = F_2$  that maximizes  $|\mathcal{U}_{F_1 \cup S}(S)|$  over all sets  $S$ , with  $|S| = k$  and  $F_1 \cap S = \emptyset$ .
- (b) The objective of  $\mathcal{P}_1$  is to place a set  $F_1$  of  $k$  facilities such that the maximum possible payoff of  $\mathcal{P}_2$  is minimized. In other words, the objective of  $\mathcal{P}_1$  is to choose a set  $F = F_1$  of size  $k$  that minimizes  $\max_S |\mathcal{U}_{F \cup S}(S)|$ , where the maximum is taken over all sets  $S$ , with  $|S| = k$  and  $F \cap S = \emptyset$ .

In this paper, we consider the case when  $k = 1$ . Thus,  $\mathcal{P}_1$  will place a single facility inside  $P$ , after which  $\mathcal{P}_2$  places another facility inside  $P$ . In the next section, we characterize an optimal placement for  $\mathcal{P}_2$  and show that, given any placement of a facility by  $\mathcal{P}_1$ , an optimal strategy for  $\mathcal{P}_2$  can be computed in  $O(m+n(\log n + \log m))$  time, where  $m$  is the number of vertices of  $P$ . In Section 3, we will provide an algorithm that computes an optimal strategy for  $\mathcal{P}_1$ .

## 2 Computing an Optimal Placement for $\mathcal{P}_2$

Let  $f$  (= first) and  $s$  (= second) be the facilities in  $P$  that are placed by players  $\mathcal{P}_1$  and  $\mathcal{P}_2$ , respectively.

Given any placement of the facility  $f$  by  $\mathcal{P}_1$ , we will provide an algorithm that computes a point  $s$  that maximizes  $|\mathcal{U}_{\{f,s\}}(\{s\})|$  over all points  $s \in P$ , where  $s \neq f$ .

Consider the set  $\mathcal{U} = \{u_1, u_2, \dots, u_n\}$  of users. For each user  $u_i$ , let  $d_i$  denote the geodesic distance between  $u_i$  and  $f$ , and let  $\Gamma_i$  denote the set of points in the polygon  $P$  whose geodesic distance to  $u_i$  is at most  $d_i$  (see Figure 1).

**Observation 1.** *A user  $u_i \in \mathcal{U}$  is served by a facility  $s$  placed by  $\mathcal{P}_2$  if and only if  $s$  belongs to  $\Gamma_i$ .*

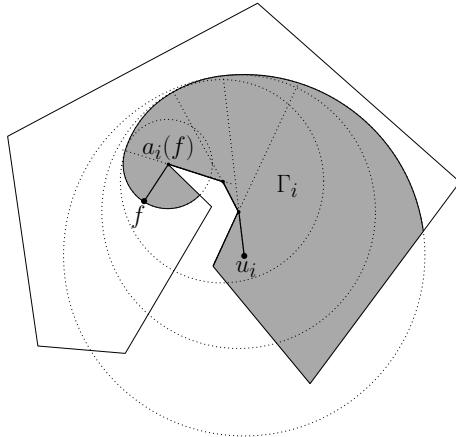
Hence, if we consider the arrangement inside  $P$  defined by  $f$  and the set of regions  $\Gamma = \{\Gamma_1, \Gamma_2, \dots, \Gamma_n\}$ , then an optimal placement for  $\mathcal{P}_2$  belongs to a cell in this arrangement having maximum depth where depth of a cell is the number of regions in  $\Gamma$  pierced by any point in that cell.

For any two points  $p_1$  and  $p_2$  in  $P$ , denote the geodesic path between  $p_1$  and  $p_2$  by  $\lambda(p_1, p_2)$ . The length of a geodesic path  $\lambda$  is denoted by  $|\lambda|$ . The *anchor* of  $f$  with respect to  $u_i$  is defined to be the last vertex on the path  $\lambda(u_i, f)$  from  $u_i$  to  $f$ ; we denote this anchor by  $a_i(f)$ . If  $f$  is visible from  $u_i$  then we define  $a_i(f) = u_i$ . Let  $C_i$  denote the circle centered at  $a_i(f)$  and passing through  $f$ . For any two points  $p_1, p_2 \in P$ , denote the line segment joining them by  $[p_1, p_2]$ .

For any anchor vertex  $a_i(f)$ , let  $l_i$  denote the line tangent to the circle  $C_i$  and passing through the point  $f$  (see Figure 2(a)). Consider the line segment  $[c, d] \subset l_i$  of maximum length that is completely contained in  $P$  and that contains  $f$ . Observe that  $[c, d]$  divides the polygon into two parts. Denote the part which contains  $u_i$  by  $P_i$ .

**Lemma 1.** *For any user  $u_i$ ,  $\Gamma_i \subseteq P_i$ .*

*Proof.* If  $a_i(f) = u_i$ , then there is nothing to prove. In the rest of the proof, we assume that  $a_i(f) \neq u_i$ . It is sufficient to prove that for any point  $q \in [c, d]$ ,  $|\lambda(u_i, q)| > |\lambda(u_i, f)|$ . Observe that  $[a_i(f), f]$  divides  $P_i$  in two parts. Without loss of generality, assume that  $d$  and  $u_i$  belong to the same sub-polygon and that  $c$  belongs to the other sub-polygon. For all points  $q \in [c, f]$ , the shortest path from  $u_i$  to  $q$  is through  $a_i(f)$ . Hence, for all points  $q \in [c, f]$ ,  $|\lambda(u_i, q)| > |\lambda(u_i, f)|$ .



**Fig. 1.** Span of the user  $u_i$

Assume there exists a point  $r$  in  $[f, d]$ , such that  $|\lambda(u_i, r)| < |\lambda(u_i, f)|$ . Let  $r \in [f, d]$  be such a point that is closest to  $f$ .

**Claim:** The set of vertices in  $\lambda(u_i, r)$  is a subset of the set of vertices in  $\lambda(u_i, f)$ .  
*Proof of Claim.* Let the vertices on the path  $\lambda(u_i, f)$  be

$$\lambda(u_i, f) = (v_1, v_2, \dots, v_j, v_{j+1}^f, \dots, v_\tau^f, f)$$

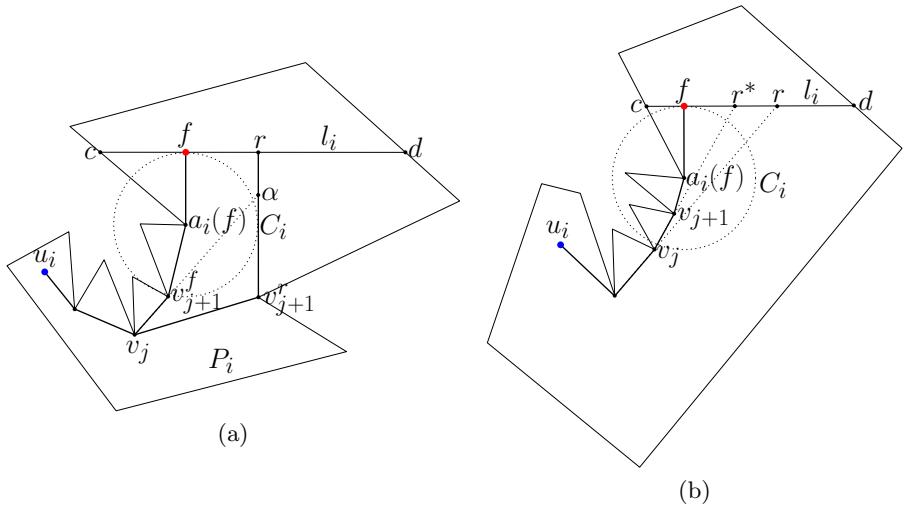
and let the vertices on the path  $\lambda(u_i, r)$  be

$$\lambda(u_i, r) = (v_1, v_2, \dots, v_j, v_{j+1}^r, \dots, v_\omega^r, r),$$

see Figure 2(a). Observe that the line joining  $v_j$  and  $v_{j+1}^f$  either intersects the line segment  $[f, r]$  or intersects some edge in  $\lambda(v_j, r)$ . If this line intersects  $[f, r]$  then that contradicts the fact that  $r$  is the closest point from  $f$  in  $[f, d]$  for which  $|\lambda(u_i, r)| < |\lambda(u_i, f)|$ . Hence, the line joining  $v_j$  and  $v_{j+1}^f$  intersects an edge of  $\lambda(v_j, r)$  (see Figure 2(a)) at a point  $\alpha$ . From the convexity properties of geodesic paths and the triangle inequality,  $|\lambda(v_j, v_{j+1}^f, \alpha)| < |\lambda(v_j, v_{j+1}^r, \alpha)|$ . This contradicts the fact that the shortest path between  $v_j$  and  $r$  is via  $v_{j+1}^r$ . Hence the claim holds.

We continue with the proof of Lemma 1. Let  $j$  be the index such that  $r$  is the intersection of  $l_i$  and the line joining the two consecutive vertices  $v_j$  and  $v_{j+1}$  of  $\lambda(u_i, f)$  (see Figure 2(b)). Denote the intersection between  $l_i$  and the line joining  $v_{j+1}$  and  $v_{j+2}$  by  $r^*$ . Observe that  $|\lambda(u_i, r^*)| < |\lambda(u_i, r)|$ . This contradicts the fact that  $r$  is the closest point from  $f$  in  $[f, d]$  for which  $|\lambda(u_i, r)| < |\lambda(u_i, f)|$ . Hence  $r$  must be the intersection of  $l_i$  and the line joining  $a_i(f)$  and  $b_i$ , where  $b_i$  is the vertex previous to  $a_i(f)$  on the path  $\lambda(u_i, f)$ . But  $|\lambda(a_i(f), f)| < |\lambda(a_i(f), r)|$ . Therefore,  $|\lambda(u_i, f)| < |\lambda(u_i, r)|$ . Hence, we arrive at a contradiction.  $\square$

For any anchor vertex  $a_i(f)$ , let  $Z_i$  denote the set of points in  $P$  which are at distance at most  $|a_i(f)f|$  from  $a_i(f)$ .



**Fig. 2.** Illustration of the proof of Lemma 1

**Observation 2.** For any two users  $u_i$  and  $u_j$ , we have  $(\Gamma_i \cap \Gamma_j) \setminus \{f\} = \emptyset$  if and only if  $(Z_i \cap Z_j) \setminus \{f\} = \emptyset$ .

*Proof.* Observe that  $Z_i \subseteq \Gamma_i$  and  $Z_j \subseteq \Gamma_j$ . Therefore, if  $(Z_i \cap Z_j) \setminus \{f\} \neq \emptyset$ , then  $(\Gamma_i \cap \Gamma_j) \setminus \{f\} \neq \emptyset$ .

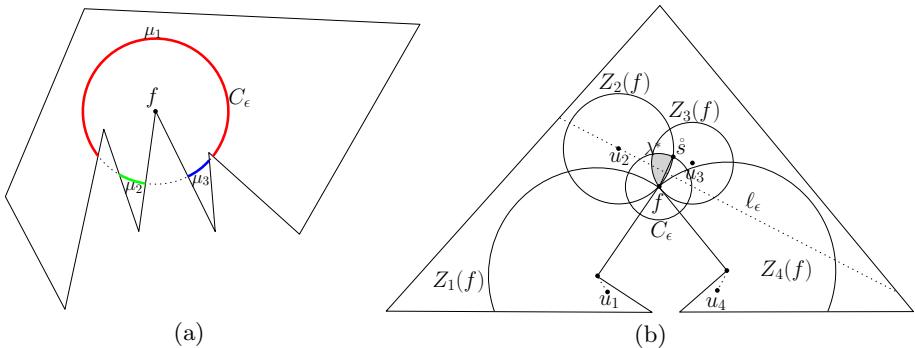
Assume that  $(Z_i \cap Z_j) \setminus \{f\} = \emptyset$ . Observe that both  $Z_i$  and  $Z_j$  contain the point  $f$ . Hence, both the circles  $C_i$  and  $C_j$  share the same tangent  $l_{ij}$  passing through  $f$ . Now  $l_{ij}$  divides  $P$  into two disjoint subpolygons  $P_i$  and  $P_j$ . Observe that  $P_i \subset \Gamma_i$  and  $P_j \subset \Gamma_j$ . Therefore,  $(\Gamma_i \cap \Gamma_j) \setminus \{f\} = \emptyset$ .  $\square$

**Observation 3.** For any placement of a facility  $f$  by  $\mathcal{P}_1$ , an optimal placement of a facility by  $\mathcal{P}_2$  is the point  $\hat{s} \neq f$  in  $P$  that pierces the maximum number of regions among  $\{Z_1, Z_2, \dots, Z_n\}$ .

*Proof.* Observe that it is enough to prove that for any subset of users  $U' \subset \mathcal{U}$ ,  $(\cap_{u_i \in U'} \Gamma_i) \setminus \{f\} \neq \emptyset$  if and only if  $(\cap_{u_i \in U'} Z_i) \setminus \{f\} \neq \emptyset$ . Now suppose  $U'$  be any subset of users for which  $(\cap_{u_i \in U'} \Gamma_i) \setminus \{f\} \neq \emptyset$ . Hence for each pair of users  $u_i, u_j \in U'$ ,  $(\Gamma_i \cap \Gamma_j) \setminus \{f\} \neq \emptyset$ . Therefore from Observation 2 for all  $u_i, u_j \in U'$ ,  $(Z_i \cap Z_j) \setminus \{f\} \neq \emptyset$ . Now all  $Z_i$  are subset of the disk passing through  $f$ . Hence  $(\cap_{u_i \in U'} Z_i) \setminus \{f\} \neq \emptyset$ . On the other hand if for any subset of users  $U'$ ,  $(\cap_{u_i \in U'} Z_i) \setminus \{f\} \neq \emptyset$  then  $(\cap_{u_i \in U'} \Gamma_i) \setminus \{f\} \neq \emptyset$  because  $Z_i \subset \Gamma_i$ . Hence the result holds.  $\square$

The arrangement of the regions  $Z_1, Z_2, \dots, Z_n$  divides  $P$  into cells. All points within the same cell pierce the same set of regions. Define the *depth* of a cell to be the number of regions pierced by any point in that cell. The cell with maximum depth contains  $f$ , because all regions  $Z_1, Z_2, \dots, Z_n$  contain  $f$ .

Consider a circle  $C_\epsilon$  with radius  $\epsilon > 0$  that is centered at  $f$ ; this circle pierces all cells containing  $f$  (see Figure 3(a)). Observe that  $C_\epsilon \cap P$  can be a set of



**Fig. 3.** Arrangement of the regions  $\{Z_1, Z_2, \dots, Z_n\}$

disjoint subsets of  $C_\epsilon$ . If  $f$  is in the interior of  $P$ , then we can choose  $\epsilon$  such that  $C_\epsilon$  is completely contained in the interior of  $P$ . If  $f$  belongs to the boundary of  $P$ , then we can choose  $\epsilon$  such that  $C_\epsilon \cap P$  is a single connected subset of  $C_\epsilon$  (see Figure 3(a) where  $C_\epsilon \cap P$  consists of three disjoint sets  $\mu_1, \mu_2$ , and  $\mu_3$ ).

Consider any optimal placement  $s$  for  $\mathcal{P}_2$ . Let  $\gamma$  be the cell that contains  $s$ . From the previous discussion, any point in  $\gamma$  acts as an optimal placement for  $\mathcal{P}_2$ . Hence, the intersection point between the boundary of  $\gamma$  and  $C_\epsilon$  is also an optimal placement. Thus, one of the optimal placements for  $\mathcal{P}_2$  belongs to the set  $\alpha_\epsilon = \{C_i \cap C_\epsilon : 1 \leq i \leq n\}$ .

Consider any optimal placement of facility  $\dot{s} \in \alpha_\epsilon$  for  $\mathcal{P}_2$ . Let  $\dot{s}$  be the intersection point between  $C_i$  and  $C_\epsilon$ . Consider the perpendicular bisector  $\ell$  of  $f$  and  $\dot{s}$ . Let  $\ell_\epsilon \subseteq \ell$  be the maximal line segment in  $P$  that contains the midpoint of the line segment joining  $f$  and  $\dot{s}$  (see Figure 3(b)).

**Observation 4.** *The line segment  $\ell_\epsilon$  passes through  $a_i(f)$ .*

*Proof.* Observe that  $[\dot{s}, f]$  is a chord of the circle  $C_i$ . The perpendicular bisector of any chord always passes through the center of the circle. Hence, the result holds.  $\square$

Note that  $\ell_\epsilon$  divides  $P$  into two sub-polygons, one containing  $f$  and the other containing  $\dot{s}$ . If  $\mathcal{P}_2$  places its facility at  $\dot{s}$ ,  $\mathcal{P}_2$  will serve the set of users  $u_i$  such that  $a_i(f)$  belongs to the sub-polygon containing  $\dot{s}$ . As  $\epsilon$  tends to 0,  $\ell_\epsilon$  tends to the line joining  $f$  and  $a_i(f)$ . Hence, for all  $a_i(f)$ , if we consider the chord passing through  $f$  and  $a_i(f)$ , then we can find an optimal placement for  $\mathcal{P}_2$ . Note that all anchor vertices are visible from  $f$ . Thus, using an angular sorting, we can find an optimal placement for  $\mathcal{P}_2$ . We obtain the following result.

**Theorem 1.** *Let  $P$  be a polygon with  $m$  vertices and let  $\mathcal{U}$  be a set of  $n$  point-users in  $P$ . Given the placement of a point  $f \in P$  by  $\mathcal{P}_1$ , a point  $s \in P$  maximizing  $\mathcal{P}_2$ 's payoff can be computed in  $O(m + n(\log m + \log n))$  time.*

*Proof.* Let  $f$  be any placement of a facility by  $\mathcal{P}_1$ . Consider the visibility region  $V_f$  of  $f$  in  $P$ , i.e., the set of points which are visible from  $f$ . Observe that

$P \setminus V_f$  consists of a set of possibly disjoint sub-polygons of  $P$ . For each such sub-polygon  $P_i$ , for all points  $q \in P_i$ , the anchor vertex on the path  $\lambda(q, f)$  will be the same. Given  $f$ , we can construct a data structure in  $O(m)$  time that can report the anchor vertex on the path  $\lambda(q, f)$ , for any query point  $q \in P$ , in  $O(\log m)$  time [6]. Using this data structure, in  $O(n \log m)$  time, we can find the set of all anchor vertices on the paths from users in  $\mathcal{U}$  to  $f$ . Once we have the list of anchor vertices, using angular sorting, we can compute the half plane passing through  $f$  which contains the maximum number of anchor vertices. Hence the result follows.  $\square$

### 3 Computing an Optimal Placement for $\mathcal{P}_1$

As before, let  $P$  be a simple polygon with  $m$  vertices and let  $\mathcal{U} = \{u_1, u_2 \dots u_n\}$  be a set of  $n$  point-users in  $P$ . We will present an algorithm that computes an optimal placement of a facility for  $\mathcal{P}_1$ .

For any placement of  $f$  by  $\mathcal{P}_1$ , let  $\nu(f) = \max_s |\mathcal{U}_{f \cup s}(\{s\})|$ , where the maximum is taken over all points  $s$  in  $P$  with  $s \neq f$ . Our objective is to find a point  $\hat{f} \in P$  which minimizes  $\nu$ ; we call such a point an *optimal placement* for  $\mathcal{P}_1$ . Observe that there are two cases:

*Case 1:*  $\hat{f}$  belongs to the boundary of  $P$ .

*Case 2:*  $\hat{f}$  is in the interior of  $P$ .

In Section 3.1, we will give an algorithm that computes an optimal placement on the boundary of  $P$  for  $\mathcal{P}_1$ . Formally, we will show how to compute a point  $f_b$  on the boundary of  $P$  such that  $\nu(f_b) = \min_f \nu(f)$ , where the minimum is taken over all points  $f$  on the boundary of  $P$ . In Section 3.2, we will give an algorithm that computes an optimal placement in the interior of  $P$  for  $\mathcal{P}_1$ .

#### 3.1 The Boundary Case

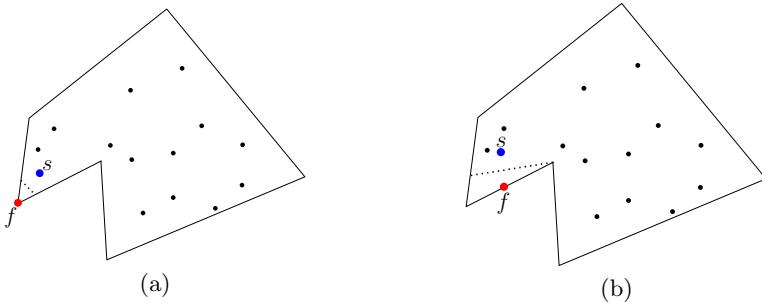
Let us begin our discussion with the following two simple observations (see Figure 4).

**Observation 5.** *For any placement  $f$  by  $\mathcal{P}_1$ , where  $f$  is on any convex vertex of  $P$ , there exists a placement  $s$  for  $\mathcal{P}_2$  such that  $\nu(f) = n$ .*

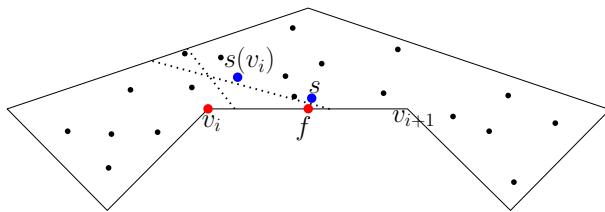
**Observation 6.** *Let  $(v_i, v_{i+1})$  be an edge of  $P$  such that at least one of  $v_i$  and  $v_{i+1}$  is a convex vertex. For any placement  $f$  by  $\mathcal{P}_1$  on the edge  $(v_i, v_{i+1})$ , there exists a placement  $s$  for  $\mathcal{P}_2$  such that  $\nu(f) = n$ .*

Hence, an optimal placement for  $\mathcal{P}_1$  must be either at a reflex vertex or on an edge  $(v_i, v_{i+1})$  for which both  $v_i$  and  $v_{i+1}$  are reflex vertices.

**Observation 7.** *Let  $(v_i, v_{i+1})$  be an edge of  $P$  such that both  $v_i$  and  $v_{i+1}$  are reflex vertices. For any placement  $f$  by  $\mathcal{P}_1$  on the edge  $(v_i, v_{i+1})$ ,  $\nu(f) \geq \nu(v_i)$  and  $\nu(f) \geq \nu(v_{i+1})$ .*



**Fig. 4.** (a) Placements for which  $\mathcal{P}_1$  gets no users



**Fig. 5.** Illustration of the proof of Observation 7

*Proof.* Let  $p$  and  $q$  be arbitrary placements by  $\mathcal{P}_1$  and  $\mathcal{P}_2$ , respectively. The perpendicular bisector of  $p$  and  $q$  divides  $P$  into two sub-polygons. Denote the sub-polygon that contains  $p$  by  $P^+(p, q)$ , and the other sub-polygon by  $P^-(p, q)$ .

Let  $f$  be any placement by  $\mathcal{P}_1$  on the edge  $(v_i, v_{i+1})$  (see Figure 5). Let  $s(v_i)$  be an optimal placement for  $\mathcal{P}_2$ , when  $\mathcal{P}_1$  places its facility at  $v_i$ . Hence,  $\nu(v_i)$  is the number of users in  $P^-(v_i, s(v_i))$ . When  $\mathcal{P}_1$  places its facility at  $f$ , there always exists a placement  $s$  by  $\mathcal{P}_2$ , such that  $s$  serves the set of users in  $P^-(v_i, s(v_i))$  (see Figure 5). Therefore,  $\nu(f) \geq \nu(v_i)$  and the claim holds.  $\square$

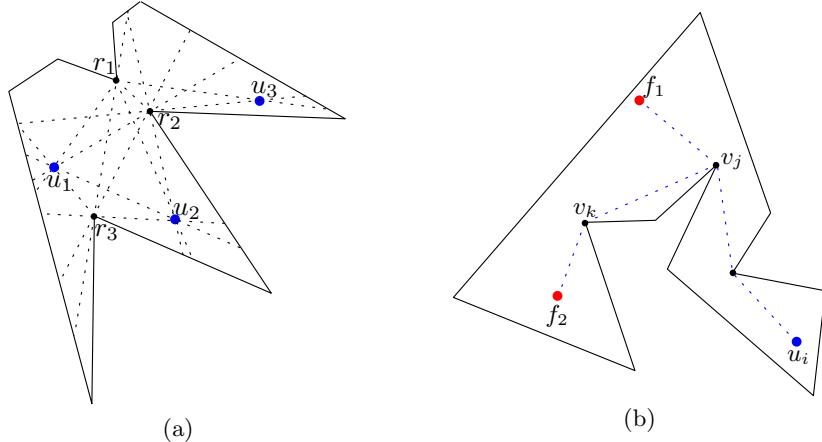
Thus, if there is optimal placement on the boundary of  $P$  for  $\mathcal{P}_1$ , that must be at a reflex vertex of  $P$ . By checking all reflex vertices, we can compute an optimal placement for  $\mathcal{P}_1$  on the boundary of  $P$  in  $O(m^2 + mn(\log m + \log n))$  time.

### 3.2 The Interior Case

In this section, we present an algorithm that computes an optimal placement for  $\mathcal{P}_1$  in the interior of the polygon  $P$ . Let  $R$  denote the set of reflex vertices of  $P$ . Consider the set  $L$  of all maximal line segments which are fully contained in  $P$  and contain at least two points from  $R \cup \mathcal{U}$  (see Figure 6(a)). The set  $L$  tessellates  $P$  into a collection of cells. Denote the tessellation by  $\Pi(P)$ .

Recall the notion of an anchor vertex defined in Section 2.

**Lemma 2.** *For any cell  $C$  in  $\Pi(P)$ , for any two points  $f_1$  and  $f_2$  in  $C$ , and for any user  $u_i$ , we have  $a_i(f_1) = a_i(f_2)$ .*



**Fig. 6.** (a) Tessellation of  $P$  (b) Illustration of the proof of Lemma 2

*Proof.* Assume there exists a user  $u_i$  whose anchor vertex  $a_i(f_1)$  on the geodesic path from  $u_i$  to  $f_1$  is different from the anchor vertex  $a_i(f_2)$  on the geodesic path from  $u_i$  to  $f_2$ .

Let  $v_j$  be the last vertex that is common to the paths  $\lambda(u_i, f_1)$  and  $\lambda(u_i, f_2)$ . Observe that one of  $a_i(f_1)$  and  $a_i(f_2)$  is not equal to  $v_j$ , because otherwise, we would have  $a_i(f_1) = a_i(f_2) = v_j$ . Assume, without loss of generality, that  $a_i(f_2) \neq v_j$  (see Figure 6(b)).

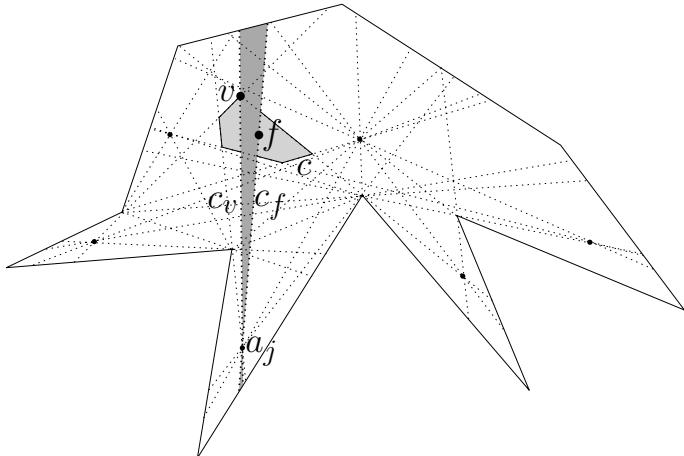
Let  $v_k$  be the vertex next to  $v_j$  on the shortest path from  $v_j$  to  $f_2$ . Observe that  $f_1$  and  $f_2$  are on different sides of the line joining  $v_j$  and  $v_k$ . Hence,  $f_1$  and  $f_2$  belong to two different cells of the tessellation  $\Pi(P)$ .  $\square$

Let  $C$  be any cell in  $\Pi(P)$ . For any user  $u_i$ , all points  $f$  in  $C$  have the same anchor vertex; denote this anchor vertex by  $a_i^C$ . Assign a weight  $w_i^C$  which is the number of shortest paths from any user  $u_j$  to any point  $f \in C$ , in which  $a_i^C$  is the anchor vertex.

Recall that a chord of  $P$  is a closed line segment whose interior is contained in the interior of  $P$  and whose endpoints are on the boundary of  $P$ . Let  $f$  be any placement by  $\mathcal{P}_1$ . Any chord passing through  $f$  divides  $P$  into two sub-polygons, which we call half polygons with respect to  $f$ . From Section 2, we know that for any placement  $f$  by  $\mathcal{P}_1$ , the maximum number of users that  $\mathcal{P}_2$  can serve, by placing one facility, is equal to  $\max_{P_f} \sum_{a_i^C \in P_f} |w_i^C|$ , where  $P_f$  is any half polygon with respect to  $f$ , i.e., the maximum number of anchor vertices in any half polygon with respect to  $f$ .

For any point  $f$  in any cell  $C$ , we define the *weighted half-space depth* of  $f$  to be  $\max_{P_f} \sum w_j^C$  such that  $a_j \in P_f$ . Observe that an optimal placement for  $\mathcal{P}_1$  in the cell  $C$  corresponds to a point with minimum weighted half-space depth.

**Lemma 3.** *One of the optimal placements for  $\mathcal{P}_1$  belongs to the set of vertices of the tessellation  $\Pi(P)$ .*



**Fig. 7.** Illustration of the proof of Lemma 3

*Proof.* Assume that none of the optimal placements for  $\mathcal{P}_1$  belongs to the set of vertices of  $\Pi(P)$ . Let  $f$  be any optimal placement for  $\mathcal{P}_1$ . Suppose  $f$  belongs to the cell  $C \in \Pi(P)$ . Let  $v$  any vertex of this cell. Let  $\delta$  be the payoff of  $\mathcal{P}_1$ , when  $\mathcal{P}_1$  places a facility at  $v$ , and assume that  $\delta$  is less than the optimal payoff of  $\mathcal{P}_1$ . Then there exists a half-polygon  $P_v$  bounded by a chord  $c_v$ , which contains  $n - \delta$  users. With out loss of generality, we may assume that  $c_v$  is passing through some anchor vertex  $a_j$ . Since  $a_j$  is visible from  $v$ ,  $a_j$  is also visible from  $f$ . Consider the chord  $c_f$  passing through  $f$  and  $a_j$  (see Figure 7). Consider the half polygon  $P_f$  bounded by  $c_f$ . Observe that  $P_v \setminus P_f = \emptyset$ , because otherwise,  $v$  and  $f$  belong to different cells of  $\Pi(P)$ . It follows that the claim holds.  $\square$

Since the cardinality of  $R \cup \mathcal{U}$  is at most  $n + m$ , the number of cells and the number of vertices in the tessellation  $\Pi(P)$  is  $O((n + m)^4)$ . For each vertex, we can check the optimal payoff of  $\mathcal{P}_1$  in  $O(m + n(\log n + \log m))$  time. Hence, we have proved the following result.

**Theorem 2.** *Let  $P$  be a polygon with  $m$  vertices and let  $\mathcal{U}$  be a set of  $n$  point-users in  $P$ . An optimal placement of a facility for  $\mathcal{P}_1$  can be computed in polynomial time.*

## 4 Conclusion

We have considered the Discrete Voronoi Game for a Simple Polygon  $P$ . The game consists of two players  $\mathcal{P}_1$  and  $\mathcal{P}_2$ , and a finite set of users in a simple polygon  $P$ . Initially,  $\mathcal{P}_1$  places one facility in  $P$ , after which  $\mathcal{P}_2$  places another facility in  $P$ . Each user is then assigned to one of the facilities according to the nearest neighbor rule, where distances are measured using the geodesic distance in  $P$ . We have shown that an optimal strategy for  $\mathcal{P}_2$ , given any placement of

$\mathcal{P}_1$ , can be found in  $O(m + n(\log m + \log n))$  time, and an optimal strategy for  $\mathcal{P}_1$  can be found in polynomial time.

There are many open problems in this area. Obtaining an algorithm to find an optimal placement for  $\mathcal{P}_1$  and  $\mathcal{P}_2$ , where each of them places  $k > 1$  facilities is a problem that remains to be solved. Another variant of the game where the two players place  $k > 1$  facilities alternately is also an interesting problem to study.

## References

1. Ahn, H.-K., Cheng, S.-W., Cheong, O., Golin, M.J., van Oostrum, R.: Competitive facility location: the Voronoi game. *Theor. Comput. Sci.* 310(1-3), 457–467 (2004)
2. Bandyapadhyay, S., Banik, A., Das, S., Sarkar, H.: Voronoi game on graphs. In: Ghosh, S.K., Tokuyama, T. (eds.) WALCOM 2013. LNCS, vol. 7748, pp. 77–88. Springer, Heidelberg (2013)
3. Banik, A., Bhattacharya, B.B., Das, S.: Optimal strategies for the one-round discrete Voronoi game on a line. *Journal of Combinatorial Optimization*, 1–15 (2012)
4. Bhattacharya, B.B., Nandy, S.C.: New variations of the maximum coverage facility location problem. *European Journal of Operational Research* 224(3), 477–485 (2013)
5. Cabello, S., Díaz-Báñez, J.M., Langerman, S., Seara, C., Ventura, I.: Facility location problems in the plane based on reverse nearest neighbor queries. *European Journal of Operational Research* 202(1), 99–106 (2010)
6. Guibas, L.J., Hershberger, J.: Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.* 39(2), 126–152 (1989)
7. Hakimi, S.L.: On locating new facilities in a competitive environment. *European Journal of Operational Research* 12(1), 29–35 (1983)
8. Kiyomi, M., Saitoh, T., Uehara, R.: Voronoi game on a path. *IEICE Transactions* 94-D(6), 1185–1189 (2011)
9. Teramoto, S., Demaine, E.D., Uehara, R.: The Voronoi game on graphs and its complexity. *J. Graph Algorithms Appl.* 15(4), 485–501 (2011)

# Facets for Art Gallery Problems

Sndor P. Fekete, Stephan Friedrichs,  
Alexander Kroller, and Christiane Schmidt

TU Braunschweig, IBR, Algorithms Group  
Mhlenpfrdtstr. 23, 38106 Braunschweig, Germany  
`{s.fekete,stephan.friedrichs,a.kroeller,c.schmidt}@tu-bs.de`

**Abstract.** We demonstrate how polyhedral methods of mathematical programming can be developed for and applied to computing optimal solutions for large instances of a classical geometric optimization problem with an uncountable number of constraints and variables.

The ART GALLERY PROBLEM (AGP) asks for placing a minimum number of stationary guards in a polygonal region  $P$ , such that all points in  $P$  are guarded. The AGP is NP-hard, even to approximate. Due to the infinite number of points to be guarded as well as possible guard positions, applying mathematical programming methods for computing provably optimal solutions is far from straightforward.

In this paper, we use an iterative primal-dual relaxation approach for solving AGP instances to optimality. At each stage, a pair of LP relaxations for a finite candidate subset of primal covering and dual packing constraints and variables is considered; these correspond to possible guard positions and points that are to be guarded.

Of particular interest are additional cutting planes for eliminating fractional solutions. We identify two classes of facets, based on EDGE COVER and SET COVER (SC) inequalities. Solving the separation problem for the latter is NP-complete, but exploiting the underlying geometric structure of the AGP, we show that large subclasses of fractional SC solutions cannot occur for the AGP. This allows us to separate the relevant subset of facets in polynomial time.

Finally, we characterize all facets for finite AGP relaxations with coefficients in  $\{0, 1, 2\}$ . We demonstrate the practical usefulness of our approach with improved solution quality and speed for a wide array of large benchmark instances.

**Keywords:** Art Gallery Problem, geometric optimization, algorithm engineering, set cover polytope, solving NP-hard problem instances to optimality.

## 1 Introduction

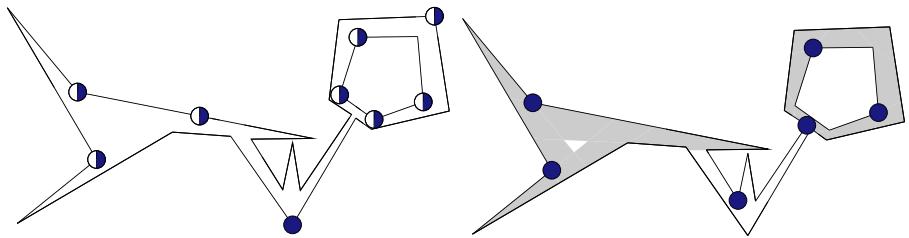
The ART GALLERY PROBLEM (AGP) is one of the classical problems of geometric optimization: given a polygonal region  $P$  with  $n$  vertices, find as few stationary guards as possible, such that any point of the region is visible by one of the guards. As first proven by Chvatal [1] and then shown by Fisk [2]

in a beautiful and concise proof (which is highlighted in the shortest chapter in “Proofs from THE BOOK” [3]),  $\lfloor \frac{n}{3} \rfloor$  guards are sometimes necessary and always sufficient when  $P$  is a simple polygon. Worst-case bounds of this type are summarized under the name “Art-Gallery-type theorems”, and used as a metaphor even for unrelated problems; see O’Rourke [4] for an early overview, and Urrutia [5] for a more recent survey.

Algorithmically, the AGP is closely related to the SET COVER (SC) problem; it is NP-hard, even for a simply connected polygonal region  $P$  [6]. There are, however, two differences to a discrete SC problem. On the one hand, it is well known that geometric variants of problems may be easier to solve or approximate than their discrete, graph-theoretic counterparts, so it is natural to explore ways to exploit the geometric nature of the AGP; on the other hand, the AGP is far from being discrete, as both the set to be covered (all points in  $P$ ) as well as the covering family (all star-shaped subregions around some point of  $P$ ) usually are uncountably infinite.

It is natural to consider more discrete versions of the AGP. Ghosh [7] showed that restricting possible guard positions to the  $n$  vertices, i.e., the AGP with vertex guards, allows an  $O(\log n)$ -approximation algorithm of complexity  $O(n^5)$ ; conversely, Eidenbenz et al. [8] showed that for a region with holes, finding an optimal set of vertex guards is at least as hard as SC, so there is little hope of achieving a better approximation guarantee than  $\Omega(\log n)$ . While these results provide tight bounds in terms of approximation, they do by no means close the book on the arguably most important aspect of mathematical optimization: combining structural insights with powerful mathematical tools in order to achieve provably optimal solutions for instances of interesting size. Moreover, even a star-shaped polygon may require a large number of vertex guards, so general AGP instances may have significantly better solutions than the considerably simpler discretized version with vertex guards.

Computing optimal solutions for general AGP instances is not only relevant from a theoretical point of view, but has also gained in practical importance in the context of modeling, mapping and surveying complex environments, such as in the fields of architecture or robotics and even medicine, which are seeking to exploit the ever-improving capabilities of computer vision and laser scanning. Amit, Mitchell and Packer [9] have considered purely combinatorial primal and dual heuristics for general AGP instances. Only very recently have researchers begun to combine methods from integer linear programming with non-discrete geometry in order to obtain optimal solutions. As we showed in [10], it is possible to combine an iterative primal-dual relaxation approach with structures from computational geometry in order to solve AGP instances with unrestricted guard positions; this approach is based on considering a sequence of primal and dual subproblems, each with a finite number of primal variables (corresponding to guard positions) and a finite number of dual variables (corresponding to “witness” positions). Couto et al. [11,12,13] used a similar approach for the AGP with vertex guards. Due to space limitations, we omit a detailed discussion of the abundant work on the AGP. Highly relevant is the paper by Balas and Ng [14]



**Fig. 1.** An optimal fractional solution of value 5 without (left) and an optimal integer solution of value 6 with cutting planes (right). Circles show guards, fill-in indicates fractional amount. Cutting planes enforce at least two guards in the left and three in the right area, both marked in grey.

on the discrete SC polytope, which describes all its facets with coefficients in  $\{0, 1, 2\}$ .

**Formal Description.** We consider a polygonal region  $P$  with  $n$  vertices that may have holes, i.e., that does not have to be simply connected. For a point  $p \in P$ , we denote by  $\mathcal{V}(p)$  the *visibility polygon* of  $p$  in  $P$ , i.e., the set of all  $q \in P$ , such that the straight-line connection  $\overline{pq}$  lies completely in  $P$ .  $P$  is *star-shaped* if  $P = \mathcal{V}(p)$  for some  $p \in P$ . The set of all such points is the *kernel* of  $P$ . For a set  $S \subseteq P$ ,  $\mathcal{V}(S) := \cup_{p \in S} \mathcal{V}(p)$ . A set  $C \subseteq P$  is a *guard cover*, if  $\mathcal{V}(C) = P$ . The AGP asks for a guard cover of minimum cardinality  $c$ ; this is the same as covering  $P$  by a minimum number of star-shaped sub-regions of  $P$ . Note that Chvátal's Watchman Theorem [15] guarantees  $c \leq \lfloor \frac{n}{3} \rfloor$ .

**Our Results.** In this paper, we extend and deepen our recent work on iterative primal-dual relaxations, by proving a number of polyhedral properties of the resulting AGP polytopes. We provide the first study of this type, and give a full characterization of all facets with coefficients 0, 1, and 2. Remarkably, we are able to exploit geometry to prove that only a very restricted family of facets of the general SC polytope will typically have to be used as cutting planes for removing fractional variables. Instead, we are able to prove that many fractional solutions only occur in intermittent SC subproblems; thus, they simply vanish when new guards or witnesses are introduced. This saves us the trouble of solving an NP-complete separation problem. Computational results illustrate greatly reduced integrality gaps for a wide variety of benchmark instances, as well as reduced solution times. Details are as follows; due to space restrictions, proofs are omitted. Related SC results are described by Balas et al. [14].

- We show how to employ cutting planes for an iterative primal-dual framework for solving the AGP. This is interesting in itself, as it provides an approach to tackling optimization problems with infinitely many constraints and variables. The particular challenge is to identify constraints that remain valid for any choice of infinitely many possible primal and dual variables, as we are not solving one particular IP, but an iteratively refined sequence.

- Based on a geometric study of the involved SC constraints, we characterize all facets of involved AGP polytopes that have coefficients in  $\{0, 1, 2\}$ . In the SC setting, these facets are capable of cutting off fractional solutions, but the separation problem is NP-complete. We use geometry to prove that only some of these facets are able to cut off fractional solutions in an AGP setting under reasonable assumptions, allowing us to solve the separation problem in polynomial time.
- We provide a class of facets based on EDGE COVER (EC) constraints.
- We demonstrate the practical usefulness of our results by showing greatly improved solution speed and quality for a wide array of large benchmarks.

## 2 Mathematical Programming Formulation and LP-Based Solution Procedure

Let  $P$  be a polygon and  $G, W \subseteq P$  sets of points for possible guard locations and *witnesses*, i. e., points to be guarded, respectively. We assume  $W \subseteq \mathcal{V}(G)$ . The AGP can be formulated as an IP denoted by  $\text{AGP}(G, W)$ :

$$\min \quad \sum_{g \in G} x_g \tag{1}$$

$$\text{s. t.} \quad \sum_{g \in G \cap \mathcal{V}(w)} x_g \geq 1 \quad \forall w \in W \tag{2}$$

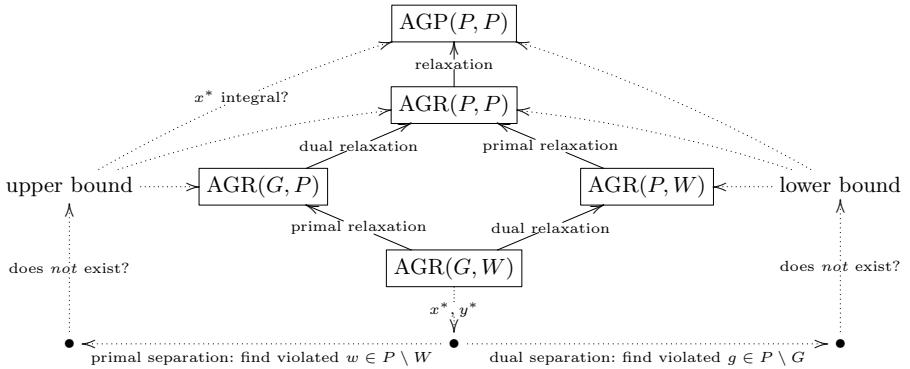
$$x_g \in \{0, 1\} \quad \forall g \in G \tag{3}$$

where Chvátal's Watchman Theorem [15] guarantees that only a finite number of variables are non-zero. The original AGP,  $\text{AGP}(P, P)$ , has uncountably many variables and constraints, so it cannot be solved directly. Thus we consider finite  $G, W \subset P$  and solve  $\text{AGP}(G, W)$ . For dual separation and to generate lower bounds, we require the LP relaxation  $\text{AGR}(G, W)$  obtained by relaxing the integrality constraint (3):

$$0 \leq x_g \leq 1 \quad \forall g \in G \tag{4}$$

The relation between a solution of  $\text{AGR}(G, W)$  and  $\text{AGR}(P, P)$  is not obvious, see Figure 2. In [10], we show that  $\text{AGR}(P, P)$  can be solved optimally for many problem instances by using finite  $G$  and  $W$ . The procedure uses primal/dual separation (i. e., cutting planes and column generation) to connect  $\text{AGR}(G, W)$  to  $\text{AGR}(P, P)$ . For some finite sets  $G$  and  $W$ , we solve  $\text{AGR}(G, W)$  using the simplex method. This produces an optimal primal solution  $x^*$  and dual solution  $y^*$  with objective value  $z^*$ . The primal is a minimum covering by guards, the dual a maximum packing of witnesses. We analyze  $x^*$  and  $y^*$  as follows:

1. If there exists a point  $w \in P \setminus W$  with  $x^*(G \cap \mathcal{V}(w)) < 1$ , then  $w$  corresponds to an inequality of  $\text{AGR}(P, P)$  that is violated by  $x^*$ . The new witness  $w$  is added to  $W$ , and the LP is re-solved. If such a  $w$  cannot be found, then  $x^*$  is optimal for  $\text{AGR}(G, P)$ , and  $z^*$  is an upper bound for  $\text{AGR}(P, P)$ .



**Fig. 2.** The AGP and its relaxations for  $G, W \subseteq P$ . Dotted arrows represent which conclusions may be drawn from the primal and dual solutions  $x^*$  and  $y^*$ .

2. If there exists a point  $g \in P \setminus G$  with  $y^*(W \cap \mathcal{V}(g)) > 1$ , then it corresponds to a violated dual inequality of  $\text{AGR}(P, P)$ . We create the LP column for  $g$  and re-solve the LP. If such a  $g$  does not exist,  $y^*$  is an optimal dual solution for  $\text{AGR}(P, W)$  and  $z^*$  is a lower bound for  $\text{AGR}(P, P)$ .

Both separation problems can be solved efficiently using the overlay of the visibility polygons of all points  $g \in G$  with  $x_g^* > 0$  (for the primal case) and all  $w \in W$  with  $y_w^* > 0$  (for the dual case), which decomposes  $P$  into a planar arrangement of bounded complexity.

Should the upper and the lower bound meet, we have an optimal solution of  $\text{AGR}(P, P)$  [10].

In this paper, we use cutting planes  $\alpha$  that must remain feasible in all iterations of our algorithm, so feasibility for  $\text{AGP}(G, W)$  is insufficient; we require  $\alpha$  not to cut off any  $x \in \{0, 1\}^{G'}$  for an arbitrary  $P \supseteq G' \supset G$ , such that  $x$  is feasible for  $\text{AGP}(G', P)$ . An LP with a set  $A$  of such additional constraints is denoted by  $\text{AGR}(G, W, A)$ , its IP counterpart by  $\text{AGP}(G, W, A)$ . Note that  $\text{AGP}(G, P)$  and  $\text{AGP}(G, P, A)$  are equivalent. By  $\text{AGP}(G, W)$ , we sometimes denote the set of its feasible solutions rather than the IP itself, as in  $\text{conv}(\text{AGP}(G, W))$ .

### 3 Set Cover Facets

In this section, we discuss a family of SC facets, and show that the underlying geometry greatly reduces their impact on the involved AGP polytopes.

#### 3.1 A Family of Facets

Let  $P$  be a polygon and  $G, W \subset P$  finite sets of guard and witness positions. Consider a finite non-empty subset  $\emptyset \subset S \subseteq W$  of witness positions; the overlay of visibility regions of  $S$  is called  $\alpha_S$ . It contains the following partition  $P =$

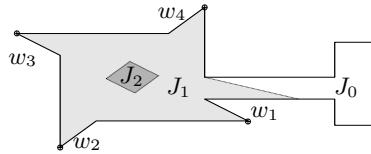
$J_0 \dot{\cup} J_1 \dot{\cup} J_2$ , cf. Fig. 3; this is analogous to what Balas and Ng [14] did for the SC polytope.

1.  $J_2 := \{g \in P \mid S \subseteq \mathcal{V}(g)\}$ , the set of positions that cover all of  $S$ .
2.  $J_0 := \{g \in P \mid \mathcal{V}(g) \cap S = \emptyset\}$ , the set of positions that see none of  $S$ .
3.  $J_1 := P \setminus (J_2 \cup J_0)$  the set of positions that cover a non-trivial subset of  $S$ .

Every feasible solution of the AGP has to cover  $S$ . Thus, it takes one guard in  $J_2$ , or at least two guards in  $J_1$  to cover  $S$ . For any  $G$ , this induces the following constraint (5); for the sake of simplicity, we will also refer to this by  $\alpha_S$ .

$$\sum_{g \in J_2 \cap G} 2x_g + \sum_{g \in J_1 \cap G} x_g \geq 2 \quad (5)$$

In the context of our iterative algorithm,  $\alpha_S$  is represented by  $J_0$ ,  $J_1$  and  $J_2$ , independent of a specific set  $G$ ; any guard  $g \in J_i$  in current or future iterations simply gets the coefficient  $\alpha_S(g) = i$ .



**Fig. 3.** Polygon and witness selection  $S = \{w_1, w_2, w_3, w_4\}$ . Guards located in  $J_2$  can cover all of  $S$ , and those in  $J_1$  some part of it, while those in  $J_0$  cover none of  $S$ .

Sufficient coverage of  $S$  is necessary for sufficient coverage of  $P$ , so (5) is valid for any  $x \in \{0, 1\}^G$  that is feasible for  $\text{AGP}(G, P)$ . However, covering  $S$  may require more than two guards in  $J_1$ , so (5) does not always provide a supporting hyperplane of  $\text{conv}(\text{AGP}(G, W))$ .

It is easy to see that for  $|S| \leq 2$ , (5) only yields constraints that are fulfilled by all feasible solutions of  $\text{AGR}(G, W)$ . Thus, we consider  $|S| \geq 3$ .

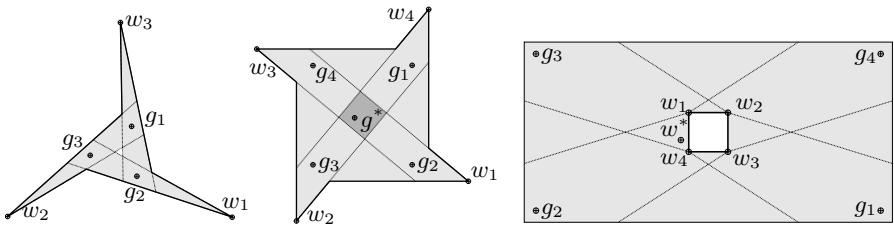
In order to show when (5) defines a facet of  $\text{conv}(\text{AGP}(G, W))$ , we need to apply a result of [14] to the AGP setting.

**Lemma 1.** *Let  $P$  be a polygon and  $G, W \subset P$  finite sets of guard and witness positions. Then  $\text{conv}(\text{AGP}(G, W))$  is full-dimensional, if and only if*

$$\forall w \in W : \quad |\mathcal{V}(w) \cap G| \geq 2 \quad (6)$$

We require more terminology adapted from [14]. Two guards  $g_1, g_2 \in J_1$  are a *2-cover* of  $\alpha_S$ , if  $S \subseteq \mathcal{V}(\{g_1, g_2\})$ . The *2-cover graph* of  $G$  and  $\alpha_S$  is the graph with nodes in  $J_1 \cap G$  and an edge between  $g_1$  and  $g_2$  iff  $g_1, g_2$  are a 2-cover of  $\alpha_S$ . In addition, we have  $T(g) = \{w \in \mathcal{V}(g) \cap W \mid \mathcal{V}(w) \cap G \cap (J_0 \setminus \{g\}) = \emptyset\}$ .

**Theorem 1.** *Given a polygon  $P$  and finite  $G, W \subset P$ , let  $\text{conv}(\text{AGP}(G, W))$  be full-dimensional and let  $\alpha_S$  be as defined in (5), such that  $S$  is maximal, i. e., there is no  $w \in W \setminus S$  with  $\mathcal{V}(w) \subseteq \mathcal{V}(S)$ . Then the constraint induced by  $\alpha_S$  defines a facet of  $\text{conv}(\text{AGP}(G, W))$ , if and only if:*



**Fig. 4.**  $P_3^2$  (left) and two attempts for  $P_4^3$  (middle and right). In the left case, Ineq. (5) enforces using two guards instead of three  $\frac{1}{2}$ -guards. The attempts for  $P_4^3$  are star-shaped (middle) or invalid (right, at  $w^*$ , as  $x_{g_1} = \dots = x_{g_4} = \frac{1}{3}$ ).

1. Every component of the 2-cover graph of  $\alpha_S$  and  $G$  has an odd cycle.
2. For every  $g \in J_0 \cap G$  such that  $T(g) \neq \emptyset$  there exists either
  - (a) some  $g' \in J_2 \cap G$  such that  $T(g) \subseteq \mathcal{V}(g')$ ;
  - (b) some pair  $g', g'' \in J_1 \cap G$  such that  $T(g) \cup S \subseteq \mathcal{V}(g') \cup \mathcal{V}(g'')$ .

### 3.2 Geometric Properties of $\alpha_S$

It is easy to construct SC instances for any choice of  $|S| \geq 3$ , such that the SC version of  $\alpha_S$  cuts off a fractional solution, cf. [14]. In general, finding  $\alpha_S$  is NP-complete. But in the following, we show that in an AGP setting, only  $\alpha_S$  with  $|S| = 3$  actually plays a role in cutting off fractional solutions under reasonable assumptions, allowing us to separate it in polynomial time.

**Lemma 2.** Let  $P$  be a polygon,  $G, W \subset P$  finite sets of guard and witness positions and  $\emptyset \subset S \subseteq W$ . If every guard in  $J_1 \cap G$  belongs to some 2-cover of  $\alpha_S$  and  $S$  is minimal for  $G$ , i.e., there is no proper subset  $T \subset S$  such that  $\alpha_T$  and  $\alpha_S$  induce the same constraint for  $G$ , the matrix of  $\text{AGP}(G, S)$  contains a permutation of the full circulant of order  $k = |S|$ , which is

$$C_k^{k-1} = \begin{pmatrix} 0 & 1 & \cdots & 1 \\ 1 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 1 & \cdots & 1 & 0 \end{pmatrix} \in \{0, 1\}^{k \times k} \quad (7)$$

Lemma 2 holds, because the 2-cover property holds iff no guard's coefficient in  $\alpha_S$  can be reduced without turning Inequality (5) invalid [14]. As  $S$  is minimal, removing  $w$  from  $S$  must increase coefficients, i.e., relocate a guard  $g \in J_1 \cap G$  to  $J_2$ . So  $\mathcal{V}(g) \cap S = S \setminus \{w\}$ . Such a guard exists for every  $w \in S$ .

This motivates a formal definition of a polygon corresponding to  $C_k^{k-1}$ .

**Definition 1 (Full Circulant Polygon).** A polygon  $P$  along with  $G(P) = \{g_1, \dots, g_k\} \subset P$  and  $W(P) = \{w_1, \dots, w_k\} \subset P$  for  $3 \leq k \in \mathbb{N}$  is called Full Circulant Polygon or  $P_k^{k-1}$ , if

$$\forall 1 \leq i \leq k : \quad \mathcal{V}(g_i) \cap W(P) = W(P) \setminus \{w_i\} \quad (8)$$

$$\forall w \in P : \quad |\mathcal{V}(w) \cap G(P)| \geq k - 1 \quad (9)$$

We may refer to  $G(P)$  and  $W(P)$  by just  $G$  and  $W$  respectively.

Note that  $P_k^{k-1}$  is defined such that the full circulant  $C_k^{k-1}$  completely describes the visibility relations between  $G$  and  $W$ . This implies that the optimal solution of  $\text{AGR}(G, W)$  is  $\frac{1}{k-1} \cdot \mathbf{1}$ , with cost  $\frac{k}{k-1}$ . It is feasible for  $\text{AGR}(G, P_k^{k-1})$  by Property (9), as any point  $w \in P_k^{k-1}$  is covered by at least  $(k-1) \cdot \frac{1}{k-1} = 1$ .

Figure 4 captures construction attempts for models of  $C_k^{k-1}$ .  $P_3^2$  exists, but for  $k \geq 4$ , the polygons are either star-shaped or not full circulant. If they are star-shaped, the optimal solution is to place one guard within the kernel. If they are not full circulant polygons, the optimal solution of  $\text{AGR}(G, W)$  is infeasible for  $\text{AGR}(G, P)$  and the current fractional solution is intermittent, i.e., cut off in the next iteration. Both cases eliminate the need for a cutting plane, and we may avoid the NP-complete separation problem by restricting separation to  $k = 3$ .

In the following we prove that  $P_k^{k-1}$  is star-shaped for  $k \geq 4$ . We start with Lemma 3, which shows that any pair of guards in  $G$  is sufficient to cover  $P_k^{k-1}$ .

**Lemma 3.** *Let  $P_k^{k-1}$  be a full circulant polygon. Then  $P_k^{k-1}$  is the union of the visibility polygons of any pair of guards in  $G(P_k^{k-1}) = \{g_1, \dots, g_k\}$ :*

$$\forall 1 \leq i < j \leq k : \quad P_k^{k-1} = \mathcal{V}(g_i) \cup \mathcal{V}(g_j) \quad (10)$$

The next step is Lemma 4, which restricts the possible structure of  $P_k^{k-1}$ .

**Lemma 4.** *Let  $P_k^{k-1}$  be a full circulant polygon with  $G(P_k^{k-1}) = \{g_1, \dots, g_k\}$ . Suppose  $k \geq 4$ . Then  $P_k^{k-1}$  has no holes.*

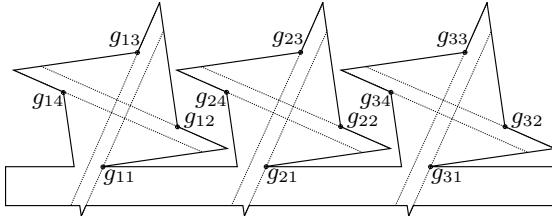
$k \geq 4$  is tight: a triangle with a concentric triangular hole is an example of  $P_3^2$ , with guards in the outside corners, and witnesses on the inside edges.

We require one final lemma before proceeding to the main Theorem 2.

**Lemma 5.** *Consider two disjoint non-empty convex polygons, described as the intersection of half-spaces:  $P_1 = \bigcap_{i=1, \dots, n} \mathcal{H}_i$  and  $P_2 = \bigcap_{i=n+1, \dots, n+m} \mathcal{H}_i$ . Then some  $\mathcal{H}_i$ ,  $1 \leq i \leq n+m$  separates  $P_1$  and  $P_2$ .*

**Theorem 2.** *A full circulant polygon  $P_k^{k-1}$  with  $k \geq 4$  is star-shaped.*

Note that Theorem 2 does not rule out situations in which  $P_k^{k-1}$  is part of a larger polygon, as shown in Figure 5. This example has no integrality gap; placing at least five copies of  $P_4^3$  around an appropriate central subpolygon with a hole can actually create one. However, such cases are much harder to come by, making these facets a lot less useful for cutting off fractional solutions; we demonstrate this in our experimental section.



**Fig. 5.** Three instances of  $P_4^3$  embedded into a larger polygon. Setting all guards to  $\frac{1}{3}$  is feasible and optimal, even though no guard is placed in any of the  $P_4^3$  kernels.

### 3.3 All Art Gallery Facets with Coefficients 0, 1, 2

Balas and Ng [14] identified all SC facets with coefficients in  $\{0, 1, 2\}$ ; for finite  $G, W \subset P$ ,  $\text{AGP}(G, W)$  is also an SC instance. Thus, all AGP facets with these coefficients must be among those facets. This includes three trivial facet classes, the constraints and the conditions under which they are facet-defining are easily translated into AGP terms; however, they are all satisfied by any feasible solution of  $\text{AGR}(G, W)$ , so they do not play a role in cutting off fractional solutions. The only non-trivial AGP facet class with coefficients in  $\{0, 1, 2\}$  is the one of type  $\alpha_S$ , as discussed above.

## 4 Edge Cover Facets

Solving  $\text{AGR}(G, W)$  for finite  $G, W \subset P$  such that no guard can see more than two witnesses is equivalent to solving fractional EC on the graph with nodes  $W$ , an edge between  $v \neq w \in W$  for each  $g \in G$  with  $\mathcal{V}(g) \cap W = \{v, w\}$ , and a loop for each  $g \in G$  with  $\mathcal{V}(g) \cap W = \{w\}$ . The fractional EC polytope is known to be half-integral [16], which can be exploited to show that fractional solutions always form odd-length cycles of  $\frac{1}{2}$ -guards.

In the conclusions of [10], we proposed a class of valid inequalities motivated by this.

A fractional optimal solution has all guard values on the cycle at  $\frac{1}{2}$ . For an odd  $k$ ,

$$\sum_{g \in \mathcal{V}(W)} x_g \geq \left\lceil \frac{k}{2} \right\rceil = \frac{k+1}{2} \quad (11)$$

separates these fractional solutions from feasible, integral solutions.

Obviously, for any choice of  $G \subset P$ , (11) does not cut off any feasible solution  $x \in \{0, 1\}^G$  of  $\text{AGP}(G, P)$ , as long as no point exists that sees more than two of these witnesses. So, analogously to the SC cuts, a cut can be kept in future iterations once it has been identified.

It is not hard to show that these are facet defining under relatively mild conditions.

**Theorem 3.** Let  $P$  be a polygon with finite sets of guard and witness positions  $G, W \subset P$ , such that  $\text{conv}(\text{AGP}(G, W))$  is full-dimensional. Let  $\overline{W} = \{w_1, \dots, w_k\} \subseteq W$  be an odd subset of  $k \geq 3$  witnesses, such that

1. No guard sees more than two witnesses in  $\overline{W}$ :

$$\forall g \in G : |\mathcal{V}(g) \cap \overline{W}| \leq 2 \quad (12)$$

2. If a guard sees two witnesses  $w_i \neq w_j \in \overline{W}$ , they are a successive pair, i.e.,  $i+1=j$  or  $i=1$  and  $j=k$ .
3. Each of the  $k$  successive pairs is seen by some  $g \in G$ .
4. No guard inside of  $\mathcal{V}(\overline{W})$  sees a witness outside of  $\overline{W}$ :

$$\forall g \in G \cap \mathcal{V}(\overline{W}) : \mathcal{V}(g) \cap W \subseteq \overline{W} \quad (13)$$

Then the constraint

$$\sum_{g \in \mathcal{V}(\overline{W}) \cap G} x_g \geq \left\lceil \frac{|\overline{W}|}{2} \right\rceil \quad (14)$$

is a facet of  $\text{conv}(\text{AGP}(G, W))$ .

## 5 Computational Experience

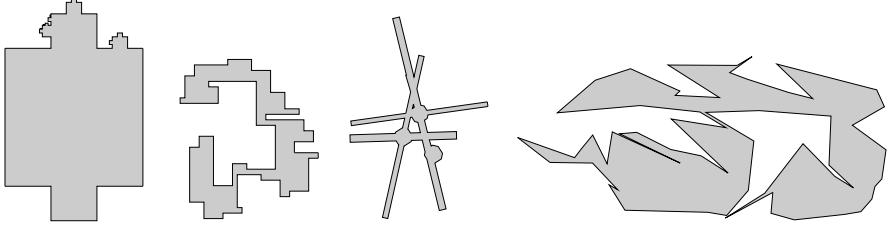
A variety of experiments on benchmark polygons demonstrates the usefulness of our cutting planes. The test algorithm is a variation of the one introduced in [10]. In each test, the sets  $G$  and  $W$  are initialized with the vertices of  $P$ , while  $A = \emptyset$ . In the primal phase, we solve  $\text{AGP}(G, W, A)$ . Should the solution be feasible for  $\text{AGP}(G, P, A)$ , we have identified an upper bound. Otherwise, there are witnesses  $W' \subseteq P \setminus W$  whose constraint is violated. In this case, the primal phase is continued and  $\text{AGP}(G, W \cup W', A)$  is solved. After an upper bound is found, the dual phase is entered. A lower bound is generated by iteratively solving the dual of  $\text{AGR}(G, W, A)$ . If the solution is feasible for the dual of  $\text{AGR}(P, W, A)$  and the cut separators do not find a violated constraint either, we have a lower bound. Otherwise, guards with violated dual constraints are added to  $G$ , violated cut conditions are added to  $A$ , and the dual phase continues. This process is repeated until the upper and the lower bound meet, or a timeout occurs.

Just as in [10], we employed four different classes of benchmark polygons.

1. Random *von Koch* polygons are inspired by Koch curves, see Fig. 6, left.
2. Random floorplan-like *Orthogonal* polygons as in Fig. 6, second polygon.
3. Random *Spike* polygons (mostly with holes) as in Fig. 6, third polygon.
4. Random non-orthogonal *Simple* polygons as in Fig. 6, fourth polygon.

Each polygon class was evaluated for different sizes  $n \in \{60, 200, 500, 1000\}$ , where  $n$  is the approximate number of vertices in a polygon.

Different combinations of cut separators were also employed. The EC-related cuts from Section 4 are referred to as *EC cuts*, while the SC-related cuts of



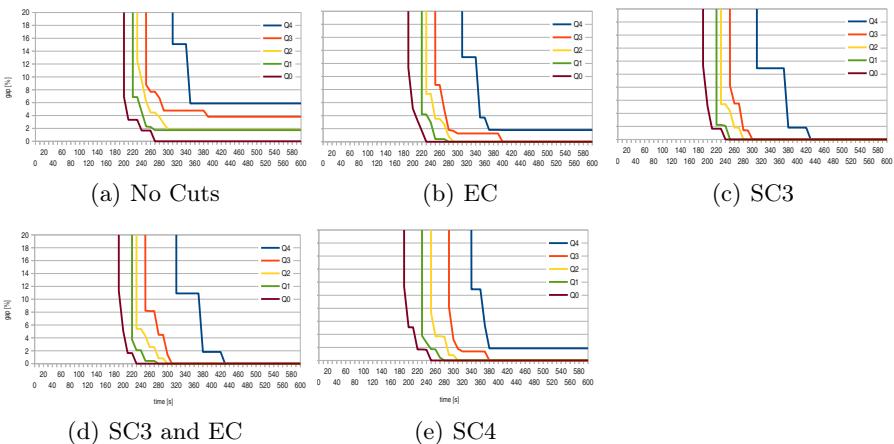
**Fig. 6.** Small *von Koch*, *Orthogonal*, *Spike* and *Simple* test polygons

Section 3 that rely on separating a maximum of  $3 \leq k$  witnesses are denoted by  $SCk$  cuts. Note that for  $k \leq m$ ,  $SCm$  cuts also include all  $SCk$  cuts.

Whenever the above algorithm separates cuts, it applies all configured cut separators and we test the following combinations: no cut separation at all,  $SC3$  cuts only,  $SC4$  cuts only,  $EC$  cuts only, and  $SC3$  and  $EC$  cuts at the same time.

In total, we have five combinations of separators, four classes of polygons and four polygon sizes; for each combination, we tested 10 different polygons. The experiments were run on 3.0 GHz Intel dual core PCs with 2 GB of memory, running 32 bit Debian 6.0.5 with Linux 2.6.32-686. Our algorithms were not parallelized, used version 4.0 of the “Computational Geometry Algorithms Library” (CGAL) and CPLEX 12.1. Each test run had a time limit of 600 s.

Below we present the relative gap over time for the five tested cut separator selections for the *von Koch*-type polygons with 1000 vertices. Fig. 7 shows the distribution of relative gaps over time for the different combination of cutting planes.  $Q_0, \dots, Q_4$  indicate the different quartiles; in particular,  $Q_0$  is the best case,  $Q_2$  the median, and  $Q_4$  the worst case. The graphs for the other test polygon types have been omitted due to space restrictions. Their analysis allows the same interpretation as ours of Fig. 7.



**Fig. 7.** Relative gap over time in IP mode for 1000-vertex *von Koch*-type polygons

Fig. 7(a) shows the relative gap over time without cut separation. After about 400s, gaps are fixed between 0% and 6%, the median gap being 2%. When applying the EC separator (Fig. 7(b)), 75% of the gaps drop to zero and the largest gap is 2%. Using the SC3 separator (Fig. 7(c)) yields an even better result in terms of both speed and relative gap. All gaps are closed, many of them earlier than with the EC separator. Combining both, see Fig. 7(d), yields a result comparable to using only SC3. Moving to the SC4 separator (Fig. 7(e)) yields a weaker performance: computation times go up, and not all gaps reach 0% within the allotted time, because separation takes longer without improving the gap. This illustrates the practical consequences of Theorem 2.

## 6 Conclusion

In this paper, we have shown how we can exploit both geometric properties and polyhedral methods of mathematical programming to solve a classical and natural, but highly challenging problem from computational geometry. This promises to pave the way for a range of practical AGP applications that have to deal with additional real-life aspects. We are optimistic that our basic approach can also be used for other geometric optimization problems related to packing and covering.

**Acknowledgments.** This work was supported by the Deutsche Forschungsgemeinschaft (DFG) under contract number KR 3133/1-1 (Kunst!).

## References

1. Chvátal, V.: A combinatorial theorem in plane geometry. *J. Combin. Theory Ser. B* 18, 39–41 (1975)
2. Fisk, S.: A short proof of Chvátal’s watchman theorem. *Journal of Combinatorial Theory (B)* 24, 374 (1978)
3. Aigner, M., Ziegler, G.M.: *Proofs from the Book*, 3rd edn. Springer (2004)
4. O’Rourke, J.: *Art Gallery Theorems and Algorithms*. International Series of Monographs on Computer Science. Oxford University Press, New York (1987)
5. Urrutia, J.: Art gallery and illumination problems. In: Sack, J.R., Urrutia, J. (eds.) *Handbook on Computational Geometry*, pp. 973–1026. Elsevier (2000)
6. Lee, D.T., Lin, A.K.: Computational complexity of art gallery problems. *IEEE Trans. Inf. Theor.* 32(2), 276–282 (1986)
7. Ghosh, S.K.: Approximation algorithms for art gallery problems in polygons and terrains. In: Rahman, M. S., Fujita, S. (eds.) *WALCOM 2010. LNCS*, vol. 5942, pp. 21–34. Springer, Heidelberg (2010)
8. Eidenbenz, S., Stamm, C., Widmayer, P.: Inapproximability results for guarding polygons and terrains. *Algorithmica* 31(1), 79–113 (2001)
9. Amit, Y., Mitchell, J.S.B., Packer, E.: Locating guards for visibility coverage of polygons. *Int. J. Comput. Geometry Appl.* 20(5), 601–630 (2010)
10. Kröller, A., Baumgartner, T., Fekete, S.P., Schmidt, C.: Exact solutions and bounds for general art gallery problems. *J. Exp. Alg.* (2012)

11. Couto, M.C., de Souza, C.C., de Rezende, P.J.: An exact and efficient algorithm for the orthogonal art gallery problem. In: SIBGRAPI 2007, pp. 87–94. IEEE Computer Society, Washington, DC (2007)
12. Couto, M.C., de Souza, C.C., de Rezende, P.J.: Experimental evaluation of an exact algorithm for the orthogonal art gallery problem. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 101–113. Springer, Heidelberg (2008)
13. Couto, M.C., de Rezende, P.J., de Souza, C.C.: An exact algorithm for minimizing vertex guards on art galleries. International Transactions in Operational Research 18, 425–448 (2011)
14. Balas, E., Ng, M.: On the set covering polytope: I. all the facets with coefficients in  $\{0, 1, 2\}$ . Math. Program. 43(1), 57–69 (1989)
15. Chvátal, V.: A combinatorial theorem in plane geometry. Journal of Combinatorial Theory, Series B 18(1), 39–41 (1975)
16. Schrijver, A.: Combinatorial Optimization - Polyhedra and Efficiency. Springer (2003)

# Hitting and Piercing Rectangles Induced by a Point Set

Ninad Rajgopal, Pradeesha Ashok, Sathish Govindarajan, Abhijit Khopkar,  
and Neeldhara Misra

Indian Institute of Science, Bangalore, India

{ninad.rajgopal,pradeesha,gsat,abhijit}@csa.iisc.ernet.in,  
mail@neeldhara.com

**Abstract.** We consider various hitting and piercing problems for the family of axis-parallel rectangles induced by a point set. Selection Lemmas on induced objects are classical results in discrete geometry that have been well studied and have applications in many geometric problems like weak epsilon nets and slimming Delaunay triangulations. Selection Lemma type results typically bound the maximum number of induced objects that are hit/pierced by a single point. First, we prove an exact result on the strong and the weak variant of the First Selection Lemma for rectangles. We also show bounds for the Second Selection Lemma which improve upon previous bounds when there are near-quadratic number of induced rectangles. Next, we consider the hitting set problem for induced rectangles. This is a special case of the geometric hitting set problem which has been extensively studied. We give efficient algorithms and show exact combinatorial bounds on the hitting set problem for two special classes of induced axis-parallel rectangles. Finally, we show that the minimum hitting set problem for all induced lines is NP-Complete.

**Keywords:** Hitting set, Selection Lemma, Centerpoint, Induced rectangles.

## 1 Introduction

Let  $P$  be a set of points in  $\mathbb{R}^d$  and let  $\mathcal{R}$  be the family of all distinct objects of a particular kind (hyperspheres, boxes, simplices, . . .), such that each object in  $\mathcal{R}$  has a distinct tuple of points from  $P$  on its boundary. For ex., in  $d = 2$ ,  $\mathcal{R}$  could be the family of  $\binom{n}{3}$  triangles such that each triangle has a distinct triple of points of  $P$  as its vertices.  $\mathcal{R}$  is called the set of all objects induced (spanned) by  $P$ . Various questions related to geometric objects induced by a point set have been studied in the last few decades. In this paper, we focus on various piercing and hitting questions on the set of induced objects  $\mathcal{R}$ . The questions are broadly classified into the following two categories:

1. What is the largest subset of  $\mathcal{R}$  that is hit/pierced by a single point?
2. What is the minimum set of points needed to hit all the objects in  $\mathcal{R}$ ?

Combinatorial results on the first category of questions are referred as *Selection Lemmas* and are well studied. A classical result in discrete geometry is the *First Selection Lemma* [9], which shows that the centerpoint [23] is present in  $\frac{n^3}{27}$  (constant fraction of) triangles induced by  $P$ . Moreover, it is known that the constant in this result is tight.

This question has also been considered for induced simplices in  $\mathbb{R}^d$ . Bárány [7] showed that there exists a point  $p \in \mathbb{R}^d$  contained in at least  $c_d \cdot \binom{n}{d+1}$  simplices induced from  $P$ . This is an important result in discrete geometry and it has been used in the construction of weak  $\epsilon$ -nets for convex objects [20]. Finding the exact constant for induced simplices in  $\mathbb{R}^d$ ,  $d \geq 3$  is considered a challenging open problem [8].

A generalization of the first selection lemma, known as *Second Selection Lemma*, considers an  $m$ -sized arbitrary subset  $\mathcal{S} \subseteq \mathcal{R}$  of distinct induced objects of a particular kind and shows that there exists a point which is contained in  $f(m, n)$  objects of  $\mathcal{S}$ . The second selection lemma has been considered for various objects like simplices, boxes and hyperspheres in  $\mathbb{R}^d$  [2, 11, 20, 24]. These results have found applications in the classical halving plane problem [2] and slimming Delaunay triangulations in 3-space [11]. For axis-parallel rectangles in  $\mathbb{R}^2$ , [11] shows a lower bound of  $\Omega(\frac{m^2}{n^2 \log^2 n})$  using induction. [24] gives an alternate proof of the same bounds using an elegant probabilistic argument and also gives an upper bound of  $O(\frac{m^2}{n^2 \log(\frac{n^2}{m})})$ . An interesting open problem mentioned in [24] is to tighten the polylogarithmic gap between these lower and upper bounds.

In this paper, we focus on induced axis-parallel rectangles in the plane. Let  $P$  be a set of  $n$  points in  $\mathbb{R}^2$  in general position i.e., no 2 points have the same  $x$  or  $y$ -coordinate and  $\mathcal{R}$  is the set of all axis-parallel rectangles induced (spanned) by  $P$  i.e., the set of  $\binom{n}{2}$  axis-parallel rectangles whose diagonal points are fixed by a pair of points from  $P$ . We obtain the following selection lemmas for axis-parallel rectangles:

- We prove a first selection lemma for axis-parallel rectangles with exact constants. We also show a *strong variant* of the first selection lemma with exact constants, where we add the constraint that the piercing point  $p \in P$ . To our knowledge, there has been no previous work on the first selection lemma for axis-parallel rectangles. Interestingly, we use the weak and strong centerpoint for rectangles [1, 6] to prove this result.
- We show bounds on  $f(m, n)$  (second selection lemma) for axis-parallel rectangles. More precisely, we show that there exists a point  $p \in \mathbb{R}^2$  that is contained in at least  $\frac{m^3}{24n^4}$  axis-parallel rectangles of  $\mathcal{S}$ . This bound is an improvement over the previous bound in [24] when  $m = \Omega(\frac{n^2}{\log^2 n})$ .

The second category of questions which we address in this paper, relates to finding a small sized hitting set for the induced objects. We consider both the algorithmic and the combinatorial bound questions on this problem.

Combinatorial bounds on the hitting set size have been studied for disks, axis-parallel rectangles and triangles [3, 4, 13, 18]. In this paper, we focus on showing combinatorial bounds on the size of the hitting set for rectangles induced by a point set. This problem is combinatorially equivalent to hitting all rectangles containing at least 2 points. Thus, this problem is a special case of the epsilon net problem where  $\epsilon = 1/n$ . Also, hitting all the induced rectangles is equivalent to hitting only those induced rectangles that do not contain any other point of  $P$ . Thus, it can be reduced to computing minimum vertex cover in the Delaunay graph w.r.t. rectangles. Bounds on the size of the independent set (complement of vertex cover) of these Delaunay graphs is well studied and is considered a challenging open problem [10, 12].

The algorithmic problem is a special case of the geometric hitting set problem. The geometric hitting set problem is NP-hard, even for simple objects like lines and unit disks [15,21] and several approximation algorithms have been proposed [5,17,22].

We show the following results on two special classes of induced axis-parallel rectangles.

- We first consider the special case of induced axis-parallel skyline rectangles. We give a simple  $O(n \log n)$  time algorithm that computes the minimum hitting set. We also give an exact combinatorial bound of  $\frac{2}{3}n$  on the size of the hitting set for induced skyline rectangles. Recently, an  $O(n^4)$  time dynamic programming based algorithm [16] was given for the more general hitting set problem for skyline rectangles (which need not be induced). Thus, our algorithm can be considered as an improvement for the case of induced skyline rectangles.
- Next, we consider the special case of induced axis-parallel slabs. We prove an exact combinatorial bound of  $\frac{3}{4}n$  on the size of the hitting set for induced axis-parallel slabs.

For most induced geometric objects, it is not known if the algorithmic problem of computing the minimum hitting set is polynomially solvable. It is known to be polynomial solvable for skyline rectangles and halfspaces. However, we show that the hitting set problem for induced lines is NP-complete by giving a reduction from Multi-colored clique. To the best of our knowledge, this is the only NP-hardness proof known for a hitting set problem in the context of objects induced by a point set. It also implies a (simpler) NP-hardness proof for the more general point line cover problem.

## 2 First Selection Lemma for Axis-Parallel Rectangles

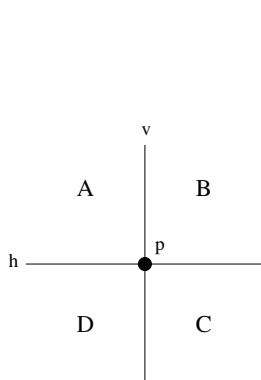
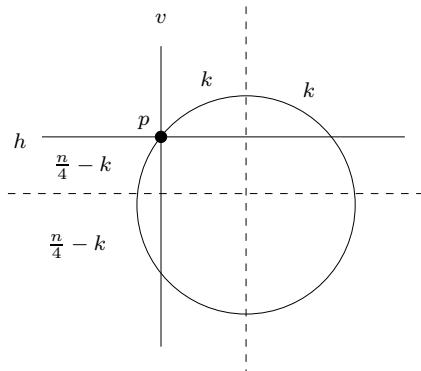
Let  $P$  be a set of  $n$  points in  $\mathbb{R}^2$  in general position i.e., no 2 points have the same  $x$  or  $y$ -coordinate. Let  $R(u, v)$  be the axis-parallel rectangle induced by  $u$  and  $v$  where  $u, v \in P$  i.e.,  $R(u, v)$  has  $u$  and  $v$  as diagonal points. Let  $\mathcal{R}$  be the set of all induced axis-parallel rectangles  $R(u, v)$  for all  $u, v \in P$ . For any point  $p$ , let  $R_p \subset \mathcal{R}$  be the set of axis-parallel rectangles that contain  $p$  and let  $f_p = |R_p|$ . Consider the quadrants formed by a horizontal and a vertical line intersecting at  $p$ .  $R_p$  consists of exactly those rectangles which are induced by a pair of points present in diagonally opposite quadrants (see figure 1).

In this section, we prove the first selection lemma for axis-parallel rectangles. We consider two variants : (1) Strong variant, where the hitting point  $p \in P$  and (2) Weak variant, where the piercing point  $p \in \mathbb{R}^2$ .

### 2.1 Strong Variant

In this section, we obtain exact bounds for  $f(n)$  where  $f(n) = \min_{P, |P|=n} (\max_{p \in P} f_p)$ .

**Theorem 1.**  $f(n) = \frac{n^2}{16}$

**Fig. 1.** Lower bound**Fig. 2.** Upper bound construction

*Proof.* Let  $p$  be the strong centerpoint of  $P$  w.r.t axis-parallel rectangles. Then any axis-parallel rectangle that contains more than  $\frac{3n}{4}$  points from  $P$  contains  $p$  [6]. We claim that  $p$  is contained in at least  $\frac{n^2}{16}$  rectangles from  $\mathcal{R}$ .

Let  $h$  and  $v$  be the horizontal and vertical lines passing through  $p$  that partition  $P$  into four quadrants as shown in figure 1. Let  $|A|$  denote  $|A \cap P|$ , for any quadrant  $A$ . If  $|A|, |C| \geq \frac{n}{4}$ , then  $p$  is contained in at least  $\frac{n^2}{16}$  rectangles from  $\mathcal{R}$ . Therefore, assume  $|A| = \frac{n}{4} - x$ . Now, there are two cases.

Case 1.  $|C| \leq \frac{n}{4}$ : W.l.o.g, assume that  $|C| = \frac{n}{4} - y$  and  $x \geq y$ . Therefore  $|B \cup D| = \frac{n}{2} + x + y$ . The value of  $f_p$  is minimized when the value of  $|B| \times |D|$  is minimized. Since  $|A| = \frac{n}{4} - x$  and there can be at most  $\frac{3n}{4}$  points on either sides of  $h$  and  $v$ , both  $B$  and  $D$  contain at least  $x$  points. Therefore,  $f_p$  is minimized when  $|B| = \frac{n}{2} + y$  and  $|D| = x$ . Then,

$$f_p \geq (\frac{n}{4} - x)(\frac{n}{4} - y) + (\frac{n}{2} + y)x \geq \frac{n^2}{16}$$

Case 2.  $|C| > \frac{n}{4}$ : Assume  $|C| = \frac{n}{4} + y$ . Therefore  $|B \cup D| = \frac{n}{2} + x - y$ . By similar reasons as in case 1, the value of  $f_p$  is minimized when  $|B| = \frac{n}{2} - y$  and  $|D| = x$ . Therefore,

$$f_p \geq (\frac{n}{4} - x)(\frac{n}{4} + y) + (\frac{n}{2} - y)x \geq \frac{n^2}{16} - 2xy + \frac{n}{4}(x + y)$$

The value of  $f_p$  is minimized at the domain boundaries and thus  $f_p \geq \frac{n^2}{16}$ .

For the upper bound, consider a set  $P$  of  $n$  points arranged uniformly along the boundary of a circle as in figure 2. Now, we claim that any point  $p \in P$  is contained in at most  $\frac{n^2}{16}$  rectangles of  $\mathcal{R}$ . W.l.o.g, let  $p$  be a point in the top left quadrant of the circle that is  $k$  points away from the topmost point in  $P$ . Let  $h$  and  $v$  be the horizontal and vertical lines passing through  $p$ .  $h$  and  $v$  divide the plane into four quadrants. Therefore  $f_p = (\frac{n}{2} - 2k)2k = nk - 4k^2$ . This value is maximized when  $k = \frac{n}{8}$ . Thus,  $f(n) \leq \frac{n^2}{16}$ .  $\square$

## 2.2 Weak Variant

In this section, we obtain tight bounds for  $f(n)$  where  $f(n) = \min_{P, |P|=n} (\max_{p \in \mathbb{R}^2} f_p)$ .

**Theorem 2.**  $f(n) = \frac{n^2}{8}$ .

*Proof.* Let  $h$  and  $v$  be the horizontal and vertical lines that bisect  $P$  and partition the plane into four quadrants. Let  $h$  and  $v$  intersect at  $p$ , which is the weak centerpoint for rectangles [1]. We claim that  $f_p \geq \frac{n^2}{8}$ .

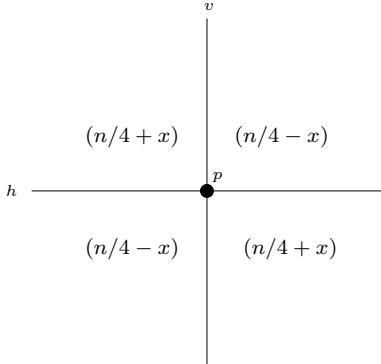


Fig. 3. Lower bound

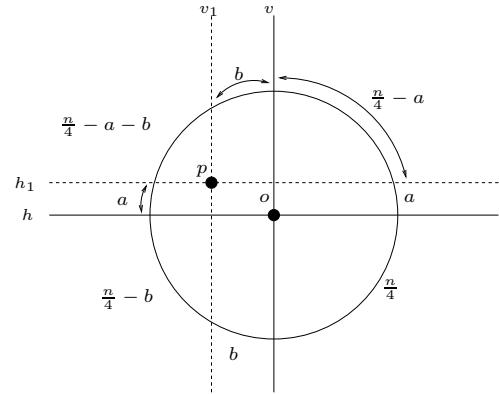


Fig. 4. Upper bound construction

Assume w.l.o.g, that the top left quadrant contains  $(\frac{n}{4} + x)$  points. Therefore, the remaining points are distributed among the three other quadrants as shown in figure 3. Then,

$$f_p = (\frac{n}{4} - x)^2 + (\frac{n}{4} + x)^2 = 2 \cdot (\frac{n^2}{16}) + 2 \cdot x^2$$

Thus,  $f_p \geq \frac{n^2}{8}$ . Therefore,  $f(n) \geq \frac{n^2}{8}$ .

For the upper bound, consider a set  $P$  of  $n$  points uniformly arranged along the boundary of a circle. Let  $h$  and  $v$  be horizontal and vertical lines that bisect  $P$ , intersecting at  $o$ . W.l.o.g, let  $p$  be any point inside the circle in the top left quadrant and let  $h_1$  and  $v_1$  be the horizontal and vertical lines passing through  $p$ . Let  $a$  be the number of points from  $P$  below  $h_1$  that is present in the top left quadrant defined by  $h$  and  $v$ . Similarly, let  $b$  be the number of points from  $P$  to the right of  $v_1$  that is present in the top left quadrant defined by  $h$  and  $v$ . The number of points in each of the four quadrants defined by  $h_1$  and  $v_1$  is as shown in figure 4.

$$f_p = (\frac{n}{4} - b + a)(\frac{n}{4} - a + b) + (\frac{n}{4} - a - b)(\frac{n}{4} + a + b) = \frac{n^2}{8} - 2(a^2 + b^2)$$

Since  $a, b \geq 0$ ,  $f_p \leq \frac{n^2}{8}$  for all points  $p \in \mathbb{R}^2$ . Therefore,  $f(n) \leq \frac{n^2}{8}$ .  $\square$

### 3 Second Selection Lemma for Axis-Parallel Rectangles in $\mathbb{R}^2$

Let  $P$  be a set of  $n$  points in  $\mathbb{R}^2$ . Let  $\mathcal{S} \subseteq \mathcal{R}$  be any set of  $m$  induced axis-parallel rectangles. In the second selection lemma, we bound the maximum number of induced rectangles of  $\mathcal{S}$  that can be pierced by a single point  $p$ . The main idea of our approach is an elegant double counting argument.

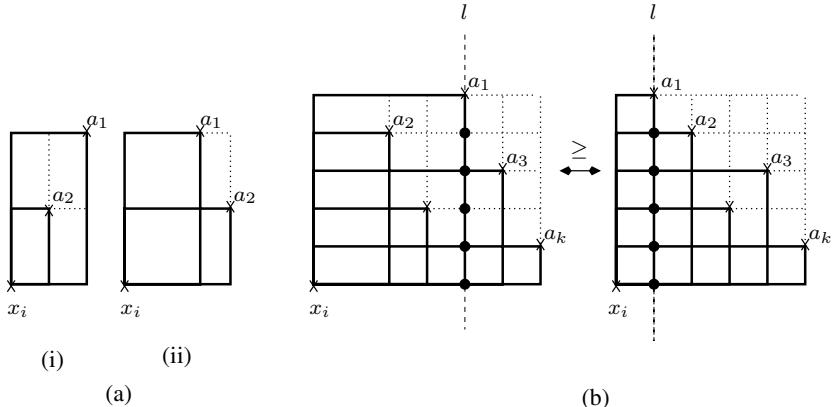
Let  $R(p, q)$  denote the rectangle induced by the points  $p$  and  $q$ .  $\mathcal{S}$  is partitioned into sets  $X_i$  as follows : any rectangle  $R(x_i, u) \in \mathcal{S}$  where  $x_i, u \in P$ , is added to the partition  $X_i$  if  $u$  is higher than  $x_i$ . Let  $P_i = \{u | R(x_i, u) \in X_i\}$ . Let  $|P_i| = |X_i| = m_i$ . Any rectangle  $R(x_i, u) \in X_i$  is placed in one of two sub-partitions,  $X'_i$  or  $X''_i$ , depending on whether  $u$  is to the right or left of  $x_i$ . Let  $|X'_i| = m'_i$  and  $|X''_i| = m''_i$ . Similarly, we partition  $P_i$  into  $P'_i$  and  $P''_i$ . Let  $\sum_{i=1}^n m'_i = m'$  and  $\sum_{i=1}^n m''_i = m''$ . The rectangles in  $X'_i$  (or  $X''_i$ ) and the points in  $P'_i$  (or  $P''_i$ ) are ordered by decreasing y-coordinate.

We construct a grid out of  $P$  by drawing horizontal and vertical lines through each point in  $P$ . Let the resulting set of grid points be  $G$  ( $P \subset G$ ), where  $|G| = n^2$ . We use the grid points in  $G$  as the candidate set of points for the second selection lemma.

Let  $J_r$  be the number of grid points in  $G$  present in any rectangle  $r \in \mathcal{S}$ . W.l.o.g consider the set of rectangles present in  $X'_i$ . We obtain a lower bound on  $\sum_{r \in X'_i} J_r$ .

$$\text{Lemma 1. } \sum_{r \in X'_i} J_r \geq \frac{(m'_i)^3}{6}.$$

*Proof.* Let  $c = \sum_{r \in X'_i} J_r$ . We prove the lemma by induction on the size of  $m'_i$ . For the base case, let  $m'_i = 2$ . There are only two ways in which the point set can be arranged, as shown in figure 5(a). It can be seen that the statement is true for the base case.



**Fig. 5.** The dotted lines represent the grid lines and the solid lines represent the rectangle edges. (a) Base cases. (b) Inductive case - the case when  $a_1$  is not the leftmost point in  $P'_i$ .

For the inductive case, assume that the statement is true for  $m'_i = k - 1$  and let  $m'_i = k$ . Let  $P'_i = \{a_1, a_2, \dots, a_k\}$ . Let  $a_1$  be the topmost point in  $P'_i$  as seen in figure 5(b) and  $l$  be the vertical line passing through  $a_1$ . We have 2 cases :

Case 1 : If  $a_1$  is the leftmost point in  $P'_i$ , then we remove  $a_1$  from  $P'_i$  and  $R(x_i, a_1)$  from  $X'_i$ . By the induction hypothesis, the lemma is true for the remaining  $k - 1$  points. On adding  $a_1$  back, we see that the line  $l$  contributes  $k$  grid points to the next rectangle in  $X'_i$ ,  $R(x_i, a_2)$ . This contribution of grid points by  $l$  becomes  $k - 1$  for the next rectangle  $R(x_i, a_3)$  and decreases by one as we move through the ordered set  $X'_i$  and it is two for  $R(x_i, a_k)$ . Thus, the total number of points contributed by  $l$  to  $c$  is given by  $\frac{k(k+1)}{2} - 1$ . The rectangle  $R(x_i, a_1)$  also contributes  $2k + 2$  to  $c$ . Thus,  $c \geq \frac{(k-1)^3}{6} + \frac{k(k+1)}{2} + (2k + 1) \geq \frac{k^3}{6}$ . Thus, the statement is true for  $m'_i = k$ .

Case 2 : If  $a_1$  is not the leftmost point, then we claim that  $c$  does not increase when we make  $a_1$  as the leftmost point by moving line  $l$  to the left. To see this, refer figure 5(b) where the grid points on  $l$  are shown as solid circles. Let  $j$  be the number of points from  $P'_i$  present to the left of  $l$ . When we make the point  $a_1$  as the leftmost by moving  $l$  to the left, we see that

- The rectangles induced by  $x_i$  and the points to the left of  $l$  have an increase in the number of grid points, which is contributed by  $l$ . Thus,  $c$  increases by  $t \leq k + (k - 1) + \dots + (k - j + 1) = \frac{j(2k+1-j)}{2}$ .
- $R(x_i, a_1)$  loses  $d = (j+2)(k+1) - 2(k+1) = j(k+1)$  points. Thus,  $c$  decreases by  $d$ .
- The number of grid points in the rectangles induced by  $x_i$  and the points to the right of  $l$  remains the same.

By a simple calculation we can see that  $d \geq t$ . Thus, when  $a_1$  is moved to the left,  $c$  does not increase. As  $a_1$  is now the leftmost point, we can apply case 1 and show that the lemma is true for  $m'_i = k$ .  $\square$

**Theorem 3.** *Let  $P$  be a point set of size  $n$  in  $\mathbb{R}^2$  and let  $\mathcal{S}$  be a set of induced rectangles of size  $m$ . If  $m = \Omega(n^{\frac{4}{3}})$ , then there exists a point  $p \in G$  which is present in at least  $\frac{m^3}{24n^4}$  rectangles of  $\mathcal{S}$ .*

*Proof.* The summation of the number of grid points present in the rectangles in  $X_i$  is given by  $\sum_{r \in X_i} J_r = \sum_{r \in X'_i} J_r + \sum_{r \in X''_i} J_r$ . Using the lower bound from Lemma 1 we have,  $\sum_{r \in X_i} J_r \geq \frac{(m'_i)^3 + (m''_i)^3}{6}$ .

Since  $\mathcal{S}$  is partitioned into the sets  $X_i$ , the summation of the number of grid points present in the rectangles in  $\mathcal{S}$  is given by

$$\sum_{r \in \mathcal{S}} J_r = \sum_{i=1}^n \sum_{r \in X_i} J_r \geq \left( \sum_{i=1}^n (m'_i)^3 + \sum_{i=1}^n (m''_i)^3 \right) / 6$$

Using Hölder's inequality in  $\mathbb{R}^n$  (generalization of the Cauchy-Schwartz inequality), we have  $\sum_{i=1}^n (m'_i)^3 \geq \frac{(m')^3}{n^2}$ . Thus, we get  $\sum_{r \in \mathcal{S}} J_r \geq \frac{(m')^3 + (m'')^3}{6n^2}$ . This sum is minimized when  $m' = m'' = \frac{m}{2}$  and thus,  $\sum_{r \in \mathcal{S}} J_r \geq \frac{m^3}{24n^2}$ .

Let  $I_g$  be the number of rectangles of  $\mathcal{S}$  containing the grid point  $g \in G$ . Now, by double counting, we have

$$\sum_{g \in G} I_g = \sum_{r \in \mathcal{S}} J_r \implies \sum_{g \in G} I_g \geq \frac{m^3}{24n^2}$$

By pigeonhole principle, there exists a grid point  $p \in G$  which is present in at least  $\frac{m^3}{24n^4}$  rectangles in  $\mathcal{S}$ .  $\square$

## 4 Hitting all Induced Rectangles

In this section, we consider the problem of hitting all induced rectangles. Specifically we consider two special cases of rectangles, namely, skylines and axis-parallel slabs.

### 4.1 Hitting Induced Skyline Rectangles

We first consider the special case of induced skyline rectangles, i.e., all the induced axis-parallel rectangles have their base extended and anchored on a common horizontal line. This is combinatorially equivalent to 3-sided axis-parallel rectangles whose base is unbounded. Let  $S(u, v)$  denote the skyline rectangle induced by  $u$  and  $v$  where  $u, v \in P$  and let  $S(P)$  denote the set of all induced skyline rectangles  $S(u, v)$  for all  $u, v \in P$ .

We now consider the problem of computing a minimum hitting set to hit  $S(P)$  for a given point set  $P$ . For any point  $u \in P$ , let  $L(u)$  denote the point in  $P$  which is the closest point to  $u$  by x-coordinate among the points which are present in the bottom-left quadrant w.r.t.  $u$ . Similarly, let  $R(u)$  denote the point in  $P$  which is the closest point to  $u$  by x-coordinate among the points which are present in the bottom-right quadrant w.r.t.  $u$ . We propose a sweep-line based algorithm to compute the minimum hitting set.

#### **Algorithm 1.** An Algorithm to hit induced skyline rectangles

```

-Set all points in  $P$  as unmarked initially.
-Consider points  $u \in P$  in decreasing order of y-coordinate.
  if  $u$  is unmarked then
    -Let  $v_1 = L(u)$  and  $v_2 = R(u)$ .
    -Include  $v_1$  and  $v_2$  in the hitting set, if they are not included already
    -Mark  $v_1$  and  $v_2$ , if they are not marked already
  else
    Continue to next point
  end if

```

**Lemma 2.** *Algorithm 1 computes a minimum hitting set for  $S(P)$  in  $O(n \log n)$  time.*

*Proof.* We first argue that the hitting set returned by above algorithm hits all the induced skyline rectangles in  $S(P)$ . Note that it is sufficient to hit induced skyline rectangles that do not contain any other point of  $P$  i.e., the hitting set forms a vertex cover in the delaunay graph of skyline rectangles. Recall that the delaunay graph for skyline rectangles has an edge  $(p, q)$  if the induced skyline rectangle  $S(p, q)$  does not contain any other point of  $P$ . First, observe that a point  $u$  has at most two edges (in the delaunay graph) to points below  $u$ , namely  $L(u)$  and  $R(u)$ . In our algorithm, either  $u$  or both  $L(u)$  and  $R(u)$  are selected (marked) in the hitting set. Thus, edges from  $u$  to vertices below  $u$  are covered and since this is true for every point  $u$ , the hitting set is a valid vertex cover.

Next, we argue that our hitting set  $H$  is in fact a minimum vertex cover (MVC). Let, if possible, there exist a different MVC  $O$ . We show that  $O$  can be transformed to  $H$  as follows: Let  $u$  be the topmost point in  $O$  that is not present in  $H$ . Assume w.l.o.g that  $v_1 = L(u)$  and  $v_2 = R(u)$  exist. Since  $u$  is not present in  $H$  (unmarked), by our algorithm,  $v_1$  and  $v_2$  will be present in  $H$ . The points  $u, v_1, v_2$  induce a triangle in the delaunay graph. Thus, at least two points in  $u, v_1, v_2$  must be selected in any vertex cover. All the three points cannot be present since then  $u$  is not needed and can be discarded. W.l.o.g, let  $O$  contain  $u$  and  $v_1$ . Now, we can replace  $u$  by  $v_2$  in the vertex cover, since  $u$  has edges only to  $v_1$  and  $v_2$  among the points below  $u$ . Performing this exchange argument from top to bottom, we transform  $O$  to  $H$ .

Sorting the points take  $O(n \log n)$  time. For a point  $u$  being considered,  $L(u)$  and  $R(u)$  can be found in  $O(\log n)$  time using range tree with fractional cascading [19]. Building the range tree takes  $O(n \log n)$  time and  $O(n \log n)$  extra space. Thus, the total running time of Algorithm 1 is  $O(n \log n)$ .  $\square$

We now obtain combinatorial bounds on the size of the hitting set.

**Theorem 4.** *Let  $P$  be a set of  $n$  points and  $S(P)$  be the family of skyline rectangles induced by  $P$ .  $S(P)$  can be hit by at most  $\frac{2n}{3}$  points of  $P$  and this bound is tight*

*Proof.* We claim that the hitting set returned by Algorithm 1 is of size at most  $\frac{2}{3}n$ . Algorithm 1 adds at most 2 points to the hitting set for every unmarked point in  $P$  and it does nothing on the selected (marked) points. Since, an unmarked point is not present in the hitting set, the size of the hitting set is at most  $\frac{2}{3}n$ .

To show the lower bound, we consider a point set  $P_1$  of three points. Let the points when sorted according to  $x$  and  $y$  co-ordinates be  $p_1, p_2, p_3$  and  $p_3, p_1, p_2$  respectively. Clearly, two points are needed to hit all skyline rectangles induced by  $P_1$ . Let  $P$  be arranged as  $\frac{n}{3}$  copies of  $P_1$  placed in the diagonal cells of a  $\frac{n}{3} \times \frac{n}{3}$  grid. Since the three skyline rectangles of a diagonal grid cell are disjoint with those of a different diagonal grid cell, at least  $\frac{2n}{3}$  hitting points are required to hit all the induced skyline rectangles.

## 4.2 Axis-Parallel Slabs

An axis-parallel slab is another special case of axis-parallel rectangle where two horizontal or vertical sides are unbounded. Thus a vertical axis-parallel slab is of the form  $[a, b] \times (-\infty, +\infty)$  and a horizontal axis-parallel slab is of the form  $(-\infty, +\infty) \times [a, b]$ . Two points  $p(x_1, y_1)$  and  $q(x_2, y_2)$  induce two axis-parallel slabs  $[x_1, x_2] \times (-\infty, +\infty)$  and  $(-\infty, +\infty) \times [y_1, y_2]$ . A family of axis-parallel slabs induced by a point set  $P$  contains all axis-parallel slabs induced by every pair of points in  $P$ .

**Theorem 5.** *Let  $P$  be a set of  $n$  points and  $S$  be the family of axis-parallel slabs induced by  $P$ .  $S$  can be hit by at most  $\frac{3n}{4}$  points of  $P$  and this bound is tight.*

*Proof.* Let  $P_x$  and  $P_y$  be ordered lists of points in  $P$  sorted according to their  $x$  and  $y$  coordinates respectively. Clearly, hitting all axis-parallel slabs of  $S$  is equivalent to hitting all empty axis-parallel slabs in  $S$ . These are exactly the vertical slabs defined by two points which are adjacent in  $P_x$  and the horizontal slabs defined by two points

which are adjacent in  $P_y$ . Therefore, at least  $\frac{n}{2}$  points from  $P_x$  as well as  $P_y$  have to be chosen as part of the hitting set,  $H$ . It will suffice to choose every alternate point starting from the first point(odd points) or starting from the second point (even points) from both  $P_x$  and  $P_y$ . W.l.o.g, assume that we add all odd points from  $P_x$  to  $H$ . Now we have the option to select either the odd points or even points from  $P_y$ . Note that  $P_y$  is a permutation of  $P_x$ . Therefore by pigeon hole principle, either the odd points from  $P_y$  or the even points from  $P_y$  contain at least  $\frac{n}{4}$  odd points from  $P_x$ . Add this set of points to  $H$ . Now  $H$  is a hitting set for  $S$  and  $|H| \leq \frac{3n}{4}$ .

To show that this bound is tight, we give a point set that needs  $\frac{3n}{4}$  points in the hitting set. Let  $P_1$  be a set of four points. Let the ordered lists of points in  $P_1$  when sorted according to the  $x$  and  $y$  coordinates be  $p_1, p_2, p_3, p_4$  and  $p_2, p_4, p_1, p_3$  respectively. Clearly three points are needed to hit all axis-parallel slabs induced by  $P_1$ . Let  $P$  contain  $\frac{n}{4}$  copies of  $P_1$  each placed along the diagonal of a  $\frac{n}{4} \times \frac{n}{4}$  grid. Since the induced axis-parallel slabs of a diagonal grid cell are disjoint with those of a different diagonal grid cell, at least  $\frac{3n}{4}$  hitting points are required to hit all the induced axis-parallel slabs.  $\square$

## 5 Hitting All the Induced Lines Is NP-Complete

Recall that the Hitting Set problem for Induced Lines is the following: given a point set  $P$ , and an integer  $k$ , we would like to determine if there is a subset  $S \subseteq P$ ,  $|S| \leq k$ , such that the set of all lines induced by  $P$  is hit by  $S$ . For points  $p$  and  $q$  in the plane, we use  $L(p, q)$  to denote the unique line passing through the points  $p$  and  $q$ . In this section, we show the following theorem.

**Theorem 6.** *The Hitting Set problem for Induced Lines is NP-complete.*

We show the NP-hardness of the hitting set for induced lines, by a reduction from Multi-Colored Clique: Given a graph  $G = (V, E)$  and a partition of  $V$  into  $k$  sets  $\{V_1, V_2, \dots, V_k\}$ , is there a clique  $C$  of size  $k$  such that  $C$  has exactly one vertex from each  $V_i$ ? This problem is well-known to be NP-Complete by an easy reduction from the classical MaxClique problem [14].

Let  $G = (V = V_1 \cup \dots \cup V_k, E)$  be an instance of Multi-Colored Clique, and let  $n := |V|$ ,  $m := |E|$ . We assume w.l.o.g, that  $G[V_i]$  is an independent set. We begin by associating a point  $p_v$  with every vertex  $v \in V$ , and a point  $p_e$  for every edge  $e \in E$ . We use  $P_V$  (respectively,  $P_E$ ) to refer to the set of points corresponding to vertices (respectively, edges). The entire point set  $P_G$ , therefore, is  $P_E \cup P_V$ , and we note that  $|P| = (m + n)$ .

The placement of the points in the plane is as follows. The points in  $P_V$  are placed in general position. The points in  $P_E$  are placed to satisfy the following properties: (a) A point  $p_e$  where  $e = (u, v)$ , is placed on the line  $L(p_u, p_v)$  and for any  $x \neq u$  or  $y \neq v$ ,  $p_e$  does not lie on  $L(p_x, p_y)$ . (b) No three points in  $P_E$  lie on a line. Further, for any  $e, f \in E$  and  $u \in V$ , there is no line that contains  $p_e, p_f \in P_E$  and  $p_u \in P_V$ . This completes the description of the placement on points in  $P_G$ . We say that  $P_G$ , defined as  $P_V \cup P_E$ , is *normalized with respect to G* if it satisfies these properties.

**Lemma 3.**  $(G, k)$ , where  $G = (V, E)$ , and  $n := |V|$ ,  $m := |E|$  is a YES-instance of Multi-Colored Clique if, and only if,  $(P_G, n + m - k)$  is a YES-instance of Hitting Set for Induced Lines.

*Proof.* For this proof, we assume (w.l.o.g), that  $k \geq 3$ . In the forward direction, let  $S \subseteq V$  be clique in  $G$  such that  $|S| = k$ . Note that  $S^* := P_E \cup (P_V \setminus \{p_v \mid v \in S\})$  is a hitting set for  $P_G$  of size  $(m + n - k)$ . Let  $e = (u, v)$ . Observe that  $L(u, v)$ , where both  $u$  and  $v$  belong to  $\{p_v \mid v \in S\}$  is hit by  $p_e \in S^*$ , and all other lines are trivially hit.

In the reverse direction, let  $S^*$  be a hitting set of size at most  $(n + m - k)$ . Let  $P_i \subseteq P_V$  denote the set  $\{p_v \mid v \in V_i\}$ . It can be argued, based on the fact that  $G[V_i]$  is independent, that  $|S^* \cap P_i| \geq |P_i| - 1$ . Next, we show that  $S^* \cap P_E = P_E$ . First note that  $|S^* \cap P_E| \geq (m - 1)$  (this is easy to derive by contradiction, given the second property of a normalized point set). Now, suppose  $|S^* \cap P_E| = (m - 1)$ . Let  $P_E \setminus S^* = \{p_e\}$ . Since  $|S^*| \leq n + m - k$ , there are at least  $(k - 1)$  parts  $P_i$  for which  $|S^* \cap P_i| = (|P_i| - 1)$  (if not, then  $|S^*| > (n + m - k)$ ). Since  $k \geq 3$ , there are at least two parts for which  $|S^* \cap P_i| = (|P_i| - 1)$ . Let the parts be  $P_a$  and  $P_b$ , and let the vertices in  $S^* \setminus P_a, S^* \setminus P_b$  be  $u$  and  $v$ , respectively. It is easy to see that at least one of  $L(p_e, p_u), L(p_e, p_v)$  is not hit by  $S^*$ , giving us the desired contradiction.

We now have that  $P_E \subseteq S^*$ . Since  $|S^*| \leq (n + m - k)$  and  $|S^* \cap P_i| \geq (|P_i| - 1)$  for all  $1 \leq i \leq k$ , it follows that  $|S^* \cap P_i| = (|P_i| - 1), 1 \leq i \leq k$ . Let  $\{P_i \setminus S^*\} = \{p_{v_i}\}$ . Let  $S = \{v_1, \dots, v_k\}$ . We claim that  $G[S]$  forms a multi-colored clique in  $G$ . It is clear that  $|S \cap V_i| = 1$  for all  $1 \leq i \leq k$ . Also,  $(v_i, v_j) \in E$  for all  $1 \leq i \neq j \leq k$ . If not, then it is not hard to see that the line  $L(p_{v_i}, p_{v_j})$  is not hit by  $S^*$ . This concludes the proof.  $\square$

**Lemma 4.** Given a graph  $G$ , a point set  $P_G$  that is normalized with respect to  $G$  can be constructed in polynomial time.

*Proof.* We begin by placing the points corresponding to  $P_V$  on a circle. Let  $E = \{e_1, \dots, e_m\}$ , where  $e_i = \{u_i, v_i\}$ . Let  $A_1$  denote the set of points where  $L(u_1, v_1)$  intersects  $L(x, y)$  for some  $x \neq u_1$  or  $y \neq v_1$  (note that  $|A_1|$  is at most  $\binom{n}{2}$ ). We place  $p_{e_1}$  arbitrarily on  $L(u_1, v_1) \setminus A_1$ . We continue this procedure iteratively. In particular, we place  $p_{e_i}$  arbitrarily on  $L(u_i, v_i) \setminus A_i$ , where  $A_i$  is the set of points where  $L(u_i, v_i)$  intersects lines formed by pairs of points in  $P_V$  and pairs of points in  $\{e_1, \dots, e_{i-1}\}$ . It is easily checked that this choice of placement is normalized with respect to  $G$ , and that  $\max |A_i| = O(n^4)$ .  $\square$

**Acknowledgements.** We would like to thank Saurabh Ray and Ajit Diwan for helpful discussions.

## References

1. Aronov, B., Aurenhammer, F., Hurtado, F., Langerman, S., Rappaport, D., Seara, C., Smorodinsky, S.: Small weak epsilon-nets. Computational Geometry 42(5), 455–462 (2009)

2. Aronov, B., Chazelle, B., Edelsbrunner, H.: Points and triangles in the plane and halving planes in space, vol. 6, pp. 435–442 (1991)
3. Aronov, B., Dulieu, M., Hurtado, F.: Witness (delaunay) graphs. *Comput. Geom.* 44(6-7), 329–344 (2011)
4. Aronov, B., Dulieu, M., Hurtado, F.: Witness gabriel graphs. *Computational Geometry* (2011)
5. Aronov, B., Ezra, E., Sharir, M.: Small-size  $\epsilon$ -nets for axis-parallel rectangles and boxes. *SIAM J. Comput.* 39(7), 3248–3282 (2010)
6. Ashok, P., Govindarajan, S., Kulkarni, J.: Small strong epsilon nets. In: CCCG, pp. 155–158 (2010)
7. Bárány, I.: A generalization of carathéodory's theorem. *Discrete Mathematics* 40(2-3), 141–152 (1982)
8. Basit, A., Mustafa, N.H., Ray, S., Raza, S.: Hitting simplices with points in  $\mathbb{R}^3$ . *Discrete & Computational Geometry* 44(3), 637–644 (2010)
9. Boros, E., Füredi, Z.: The number of triangles covering the center of an n-set. *Geometriae Dedicata* 17, 69–77 (1984)
10. Chan, T.: Conflict-free coloring of points with respect to rectangles and approximation algorithms for discrete independent set. In: Proceedings of the 2012 Symposium on Computational Geometry, pp. 293–302. ACM (2012)
11. Chazelle, B., Edelsbrunner, H., Guibas, L.J., Hershberger, J., Seidel, R., Sharir, M.: Selecting heavily covered points. *SIAM J. Comput.* 23(6), 1138–1151 (1994)
12. Chen, X., Pach, J., Szegedy, M., Tardos, G.: Delaunay graphs of point sets in the plane with respect to axis-parallel rectangles. *Random Struct. Algorithms* 34(1), 11–23 (2009)
13. Czyzowicz, J., Kranakis, E., Urrutia, J.: Dissections, cuts and triangulations. In: CCCG (1999)
14. Fellows, M.R., Hermelin, D., Rosamond, F.A., Vialette, S.: On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.* 410(1), 53–61 (2009)
15. Fowler, R.J., Paterson, M., Tanimoto, S.L.: Optimal packing and covering in the plane are np-complete. *Inf. Process. Lett.* 12(3), 133–137 (1981)
16. Grant, E., Chan, T.M.: Exact algorithms and apx-hardness results for geometric set cover. In: CCCG (2011)
17. Hochbaum, D.S., Maass, W.: Approximation schemes for covering and packing problems in image processing and vlsi. *J. ACM* 32(1), 130–136 (1985)
18. Katchalski, M., Meir, A.: On empty triangles determined by points in the plane. *Acta Mathematica Hungarica* 51(3-4), 323–328 (1988)
19. Lueker, S.: A data structure for orthogonal range queries. In: Proceedings of the 19th Annual Symposium on Foundations of Computer Science, SFCS 1978, pp. 28–34. IEEE Computer Society, Washington, DC (1978)
20. Matousek, J.: Lectures on Discrete Geometry. Springer (2002)
21. Megiddo, N., Tamir, A.: On the complexity of locating linear facilities in the plane. *Operations Research Letters* 1(5), 194–197 (1982)
22. Mustafa, N.H., Ray, S.: Improved results on geometric hitting set problems. *Discrete & Computational Geometry* 44(4), 883–895 (2010)
23. Rado, R.: A theorem on general measure. *Journal of the London Mathematical Society* 1(4), 291–300 (1946)
24. Smorodinsky, S., Sharir, M.: Selecting points that are heavily covered by pseudo-circles, spheres or rectangles. *Combinatorics, Probability & Computing* 13(3), 389–411 (2004)

# Realistic Roofs over a Rectilinear Polygon Revisited\*

Jessica Sherette<sup>1</sup> and Sang Duk Yoon<sup>2</sup>

<sup>1</sup> Department of Computer Science, University of Texas at San Antonio, USA  
*jsherett@cs.utsa.edu*

<sup>2</sup> Department of Computer Science and Engineering, POSTECH, Pohang, Korea  
*egooana@postech.ac.kr*

**Abstract.** A common task in automatically reconstructing a three dimensional city model from its two dimensional map is to compute all the possible roofs over the ground plans. A *roof* over a simple polygon in the  $xy$ -plane is a terrain over the polygon such that each face  $f$  of the terrain is supported by a plane passing through at least one polygon edge and making a dihedral angle  $\frac{\pi}{4}$  with the  $xy$ -plane [3]. This definition, however, allows roofs with faces isolated from the boundary of the polygon and local minimum edges inducing pools of rainwater. Recently, Ahn et al. [1, 2] introduced “realistic roofs” over a simple rectilinear polygon  $P$  with  $n$  vertices by imposing two additional constraints under which no isolated faces and no local minimum vertices are allowed. Their definition is, however, too restrictive that it excludes a large number of roofs with no local minimum edges. In this paper, we propose a new definition of realistic roofs corresponding to the class of roofs without isolated faces and local minimum edges. We investigate the geometric and combinatorial properties of realistic roofs and show that the maximum possible number of distinct realistic roofs over  $P$  is at most  $1.3211^m \binom{m}{\lfloor \frac{m}{2} \rfloor}$ , where  $m = \frac{n-4}{2}$ . We also present an algorithm that generates all combinatorial representations of realistic roofs.

## 1 Introduction

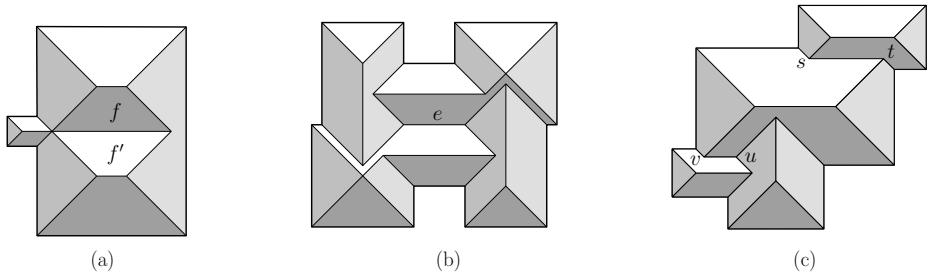
A common task in automatically reconstructing a three dimensional city model from its two dimensional map is to compute all the possible roofs over the ground plans of its buildings extensively [5,6,9,7,8,10]. For some applications, a correct or reasonable roof over a building is chosen from the candidates using some other information such as its images.

Aichholzer et al. [3] defined a *roof* over a simple (not necessarily rectilinear) polygon in the  $xy$ -plane as a terrain over the polygon such that each face of the terrain is supported by a plane passing through at least one polygon edge and making a dihedral angle  $\frac{\pi}{4}$  with the  $xy$ -plane. This definition, however, is

---

\* This research was supported by NRF grant 2011-0030044 (SRC-GAIA) funded by the government of Korea.

not tight enough that it allows roofs with faces isolated from the boundary of the polygon (Figure 1(a)) and local minimum edges (Figure 1(b)) which are undesirable for some practical reasons – for example, a local minimum edge serves as a pool of rainwater. Note that a pool of rainwater on a roof always contains a local minimum edge.



**Fig. 1.** (a) A roof with isolated faces  $f$  and  $f'$ . (b) A roof with a local minimum edge  $e$ . (c) Not a realistic roof according to Definition 1; vertex  $u$  has no adjacent vertex  $v$  that is lower than  $u$ .

### 1.1 Related Work

Brenner [6] designed an algorithm that computes all the possible roofs over a rectilinear polygon, but no polynomial bound on its running time is known. Recently, Ahn et al. [1,2] introduced “realistic roofs” over a rectilinear polygon  $P$  with  $n$  vertices by imposing two additional constraints as follows.

**Definition 1 ([1]).** A *realistic roof* over a simple rectilinear polygon  $P$  is a roof over  $P$  satisfying the following constraints.

C1. *Each face of the roof is incident to at least one edge of  $P$ .*

C2. *Each vertex  $u$  of the roof is higher than at least one of its neighboring vertices.*

They showed some geometric and combinatorial properties of realistic roofs, including a connection to the *straight skeleton* [3,4]. Consider a roof  $R^*(P)$  over  $P$  constructed by *shrinking process*, where all of the edges of  $P$  move inside, being parallel to themselves, with the same speed while moving upward along the  $z$ -axis with the same speed. Ahn et al. [1,2] showed that  $R^*(P)$  is unique, its projection on the  $xy$ -plane is the straight skeleton of  $P$ , and it is the point-wise highest realistic roof over  $P$ . From the fact that  $R^*(P)$  does not have a “valley”, we can construct another realistic roof  $R$  over  $P$  which is different to  $R^*(P)$  by adding a set of “compatible valleys” to  $R^*(P)$ . They showed that the number of realistic roofs lies between 1 and  $\lfloor \frac{m}{2} \rfloor$  where  $m = \frac{n-4}{2}$ , and presented an output sensitive algorithm generating all combinatorial representations of realistic roofs over  $P$  in  $O(1)$  time per roof, after  $O(n^4)$  preprocessing time.

## 1.2 Our Results

Constraint  $C1$  in Definition 1 is to exclude roofs with isolated faces and constraint  $C2$  is introduced to avoid pools of rainwater. However,  $C2$  is too restrictive that it also excludes a large number of roofs with no local minimum edges. For example, the roof in Figure 1 (c) is not realistic according to Definition 1 –  $u$  is a local minimum vertex – though rainwater can be drained well along  $uv$ . Therefore, Definition 1 by Ahn et al. does describe only a proper subset of “realistic” roofs.

We introduce a new definition of realistic roofs by replacing  $C2$  of Definition 1 with a relaxed one that excludes roofs with local minimum edges.

**Definition 2.** *A realistic roof over a simple rectilinear polygon  $P$  is a roof over  $P$  satisfying the following constraints.*

*C1. Each face of the roof is incident to at least one edge of  $P$ .*

*C2'. For each roof edge  $uv$ ,  $u$  or  $v$  is higher than at least one of its neighboring vertices.*

From now on, we use Definition 2 for realistic roofs unless stated explicitly. One important difference is that our realistic roofs do not have local minimum edges. Our definition corresponds to the class of roofs without isolated faces and local minimum edges exactly. This is often required for roofs in the real-world, as rainwater cannot be well drained from a local minimum along the roof surface.

Our main results are threefold:

1. We provide a new definition of “realistic roofs” corresponding to the real-world roofs and investigate geometric properties of them.
2. We show that the maximum possible number of realistic roofs over a rectilinear  $n$ -gon is at most  $1.3211^m \binom{m}{\lfloor \frac{m}{2} \rfloor}$ , where  $m = \frac{n-4}{2}$ .
3. We present an algorithm that generates all combinatorial representations of realistic roofs over a rectilinear  $n$ -gon, including roofs with *open valleys* only and roofs with *half-open valleys*. Precisely, it generates a roof with *open valleys* only in  $O(1)$  time after  $O(n^4)$  preprocessing time [1]. For each such roof, it generates  $O(1.3211^m)$  roofs with half-open valleys in time  $O(m1.3211^m)$ .

## 2 Preliminary

For a point  $p$  in  $\mathbb{R}^3$ , we denote by  $z(p)$  the  $z$ -coordinate of  $p$  and by  $\bar{p}$  the orthogonal projection of  $p$  onto the  $xy$ -plane. For a point  $p \in R$ , let  $D(p)$  be a square centered at  $\bar{p}$  with side length  $z(p)$ . For any two points  $s, t \in \mathbb{R}^2$ , let  $d_\infty(s, t)$  be  $L_\infty$  distance between  $s$  and  $t$ , and  $d_\infty(s, A) := \inf_{a \in A} d_\infty(s, a)$  for any set  $A$  in  $\mathbb{R}^2$ . We denote by  $\partial P$  the boundary of  $P$  and by  $\text{edge}(f)$  the edge of  $\partial P$  incident to a face  $f$  of a roof.

**Lemma 1.** ([1]) *Let  $R$  be a roof over a rectilinear simple polygon  $P$ . The followings hold.*

1. For any point  $p \in R$ , we have  $z(p) \leq d_\infty(\bar{p}, \partial P)$  and  $D(p) \subseteq P$ .
2. For each edge  $e$  of  $P$ , there exists a unique face  $f$  of  $R$  incident to  $e$ .
3. Every face  $f$  of  $R$  is monotone with respect to the line containing edge( $f$ ).

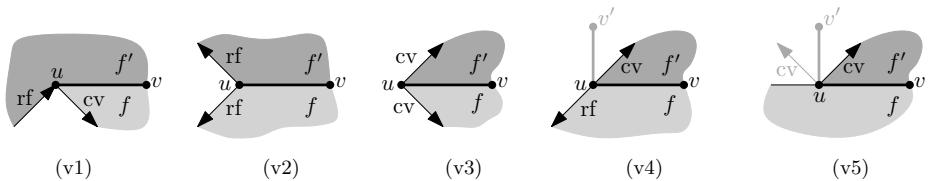
**Edge Types.** Edges of a realistic roof  $R$  over  $P$  can be classified into two groups, *convex edges* and *reflex edges*. An edge  $e$  of a roof  $R$  is called *convex* if  $R$  is locally convex along  $e$ , and an edge  $e'$  is called *reflex* if  $R$  is locally reflex along  $e'$ . Also, we will call an edge  $e$  a *valley* if  $e$  is reflex and parallel to the  $xy$ -plane, and call an edge  $e'$  a *ridge* if  $e'$  is convex and parallel to the  $xy$ -plane.

### 3 Valleys of a Realistic Roof

In this section, we investigate local structures of realistic roofs. Ahn et al. showed that realistic roofs with Definition 1 can have vertices of 5 different types for a ridge and vertices which are not incident to a valley or a ridge are degenerated forms of valleys or ridges. Since replacing constraint  $C2$  to  $C2'$  does not affect ridges, so we care about only valleys.

We define three types of valleys and describe the structures of valleys that a realistic roof can have. We call a vertex of a roof *open* if it is higher than at least one of its neighboring vertices, and *closed* otherwise. We call a valley *open* if both corners are open vertices, *half-open* if one corner is an open vertex and the other is a closed vertex, and *closed* if both corners are closed vertices. By Definition 2, a realistic roof can contain open and half-open valleys unless they make an isolated face, but it does not contain closed valleys. Ahn et al. showed that each open valley always has the same structure as *st* in Figure 1(c). More specifically, they showed that there are only 5 possible configurations at a corner of a valley of a roof because of the roof constraints such as the monotonicity of a roof, and the slope and orientations of faces as illustrated in Figure 2. They showed that a realistic roof has an open valley both corners of which are of type (v1) and oriented symmetrically along the valley. Also each corner of an open valley is connected to a reflex vertex of  $P$  by a reflex edge. We call these two reflex vertices a *candidate pair*.

In the following we investigate the structure of a half-open valley that a realistic roof can have. It is not difficult to see that the open corner is always of type



**Fig. 2.** 5 configurations around a vertex of a valley  $uv$ , where  $rf$  denotes a reflex edge and  $cv$  denotes a convex edge. Each convex or reflex edge is oriented from the endpoint with smaller  $z$ -coordinate to the other one with larger  $z$ -coordinate.

(v1) – all the others cannot have a lower neighboring vertex because of the roof constraints, that is, they are all closed. We will show that the closed corner of a realistic roof is always of type (v2). For this, we need a few technical lemmas. A proof of the following lemma can be found in the full version of the paper.

**Lemma 2.** *Let  $uv$  be a valley and  $uv'$  be a convex edge connected to  $u$ . Also, let  $\ell$  be the line in the  $xy$ -plane passing through  $\bar{v}$  and orthogonal to  $\bar{uv}$ . Then the face  $f$  incident to both  $uv$  and  $uv'$  has  $\text{edge}(f)$  in the half-plane of  $\ell$  in the  $xy$ -plane not containing  $\bar{u}$ .*

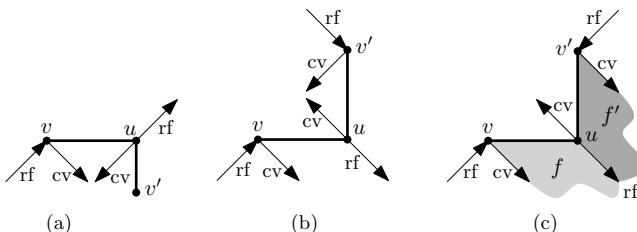
Imagine that a face  $f$  is incident to a valley  $uv$  and two convex edges one of which is incident to  $u$  and the other to  $v$ . Then both convex edges must lie in the same side of the plane containing  $uv$  and parallel to the  $z$ -axis. Since both convex edges make an angle  $45^\circ$  with  $uv$  in their projection on the  $xy$ -plane,  $f$  cannot have a ground edge, by Lemma 2, that is,  $f$  is *isolated*.

**Lemma 3.** *Let  $uv$  be a half-open valley of a realistic roof where  $u$  is closed and  $v$  is open. Then  $v$  is of type (v1) and  $u$  is of type (v2).*

*Proof.* If  $u$  is of type (v3), then one of two faces incident to  $uv$  becomes isolated by Lemma 2. If  $u$  is of type (v5), then there always is another valley  $uv'$  that is orthogonal to  $uv$  and has a closed corner at  $u$  of type (v3) as shown in Figure 2. Therefore one of faces incident to  $uv'$  is isolated.

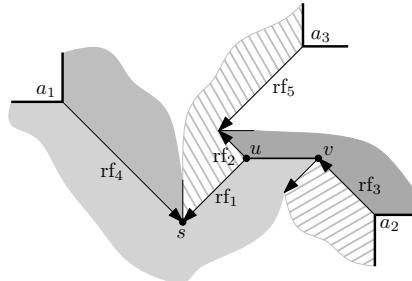
Next, assume that  $u$  is of type (v4). Then there always is another valley  $uv'$  orthogonal to  $uv$ . Therefore, we need to check two connected valleys  $uv$  and  $uv'$  simultaneously. Figure 3 illustrates all possible combinations of these two valleys. For cases (a) and (b), there is an isolated face incident to  $uv$  or  $uv'$ . For case (c), let  $f$  and  $f'$  be the faces incident to  $uv$  and  $uv'$  respectively, as shown in Figure 3(c). Let  $\ell$  ( $\ell'$  resp.) be a line in the  $xy$ -plane which contains  $\bar{uv}$  ( $\bar{uv}'$  resp.). By Lemma 2,  $\text{edge}(f)$  and  $\text{edge}(f')$  are located in opposite quadrants defined by  $\ell$  and  $\ell'$ . Then one of  $f$  and  $f'$  violate the monotonicity property (3) of Lemma 1. All valleys containing a vertex of type (v4) are invalid; the remaining closed corner is of type (v2).  $\square$

Now we are ready to describe the structure of a half-open valley. A proof of the following lemma can be found in the full version of the paper.



**Fig. 3.** Three possible combinations around a (v4) type vertex

**Lemma 4.** Let  $uv$  be a half-open valley where  $u$  is closed and  $v$  is open. Then  $uv$  is associated with three reflex vertices of  $P$  that have mutually different orientations as in Figure 4.



**Fig. 4.** A half-open valley  $uv$  must be connected to three reflex vertices  $a_1, a_2$  and  $a_3$  via 5 reflex edges. Call the vertex  $s$  which is incident to  $rf_1$  and  $rf_4$  as a *peak point* of  $uv$ .

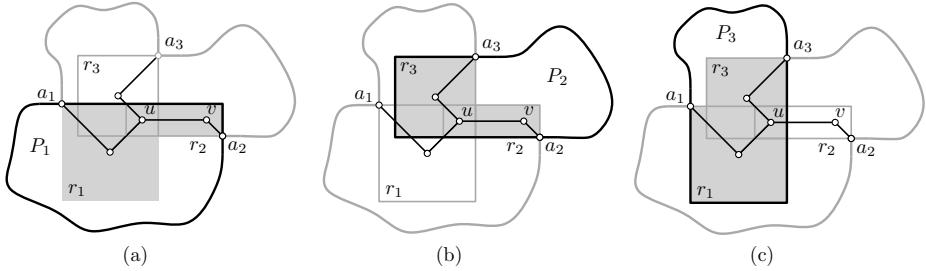
## 4 Realistic Roofs with Half-Open Valleys

**Three Reflex Vertices from a Half-Open Valley.** We investigate local and global properties of half-open valleys. From Lemma 4, we know that a half-open valley is associated with three reflex vertices that have mutually different orientations. Therefore, we need a condition to check whether chosen three reflex vertices could induce a half-open valley. Let  $a_1, a_2$  and  $a_3$  be reflex vertices which have mutually different orientations. Without loss of generality, we can assume that these vertices are arranged as in Figure 4. Let the horizontal difference of two vertices  $a_i$  and  $a_j$  be  $d_h(a_i, a_j)$  and the vertical difference be  $d_v(a_i, a_j)$ . We now consider two squares and one rectangle to determine whether these three vertices form a half-open valley. Let  $r_1$  be the square with  $a_1$  on its top left corner and side length  $d_h(a_1, a_3)$ . Let  $r_2$  be the rectangle with  $a_2$  on its bottom right corner with height  $d_v(a_1, a_2)$  and width  $d_v(a_1, a_2) + d_h(a_2, a_3)$ . Finally, let  $r_3$  be the square with  $a_3$  on its top right corner and side length  $d_v(a_2, a_3)$ .

Next, define three rectilinear subpolygons of  $P$ . Let  $P_1$  be the polygon formed by  $r_1 \cup r_2$  and the cut off portion of  $P$  below  $r_1, r_2$  and  $r_3$  (Figure 5(a)). Let  $P_2$  be the polygon formed by  $r_2 \cup r_3$  and the cut off portion of  $P$  right of  $r_1, r_2$  and  $r_3$  (Figure 5(b)). Let  $P_3$  be the polygon formed by  $r_1 \cup r_3$  and the cut off portion of  $P$  above of  $r_1, r_2$  and  $r_3$  (Figure 5(c)). We now present the following lemma, and a proof of the lemma can be found in the full version of the paper.

**Lemma 5.** Three reflex vertices  $a_1, a_2$  and  $a_3$  form a half-open valley  $uv$  if and only if  $(r_1 - a_1) \cap \partial P = \emptyset$ ,  $(r_2 - a_2) \cap \partial P = \emptyset$  and  $(r_3 - a_3) \cap \partial P = \emptyset$ .

If three reflex vertices  $a_1, a_2$  and  $a_3$  induce a half-open valley  $uv$ , we call the triple  $(a_1, a_2, a_3)$  a *candidate triple* of  $uv$ . Assume that three reflex vertices of a candidate triple are ordered as depicted in Figure 4, or the mirror image of Figure 4. Also, We will call  $r_1 \cup r_2 \cup r_3$  as the *free space* of  $uv$ .



**Fig. 5.** Dividing  $P$  into three rectilinear subpolygons,  $P_1$ ,  $P_2$  and  $P_3$ , along a half-open valley  $uv$

**Compatibility.** What we have to do next is to consider compatibilities. There are two cases: compatibility between two half-open valleys, and compatibility between an open valley and a half-open valley. We start with a lemma which states the compatibility between two open valleys.

**Lemma 6 ([1]).** *Let  $(a_1, a_2)$  and  $(a'_1, a'_2)$  be the candidate pairs for open valleys  $uv$  and  $u'v'$ , respectively.  $(a_1, a_2)$  and  $(a'_1, a'_2)$  are compatible if and only if  $\overline{C}_{a_1 a_2} \cap \overline{C}_{a'_1 a'_2} = \emptyset$ , where  $\overline{C}_{a_1 a_2} := a_1 \bar{u} \cup \bar{u}v \cup \bar{v}a_2$  and  $\overline{C}_{a'_1 a'_2} := a'_1 \bar{u}' \cup \bar{u}'v' \cup \bar{v}a'_2$ .*

Following two lemmas describe the compatibilities between two half-open valleys, and between a half-open valley and an open valley.

**Lemma 7.** *Let  $(a_1, a_2, a_3)$  and  $(a'_1, a'_2, a'_3)$  be candidate triples for two half-open valleys  $uv$  and  $u'v'$ . Then  $uv$  and  $u'v'$  are compatible if*

1. *All  $a_1, a_2$  and  $a_3$  are contained in one of  $\partial P \setminus \{a'_1, a'_2, a'_3\}$ .*
2. *Let  $P_{1a'}, P_{2a'}$  and  $P_{3a'}$  be rectilinear subpolygons of  $P$  divided by  $u'v'$  as we did before. If condition 1 is hold, then the free space of  $uv$  is contained in one of  $P_{1a'}, P_{2a'}$  and  $P_{3a'}$  that contains  $a_1, a_2$  and  $a_3$  in its boundary.*

*Proof.* Lemma 4 shows that a half-open valley must be connected to three reflex vertices via 5 reflex edges. Suppose that condition 1 does not hold. It makes some of reflex edges of  $uv$  and  $u'v'$  crossing in their projection on the  $xy$ -plane. Among crossing edges, the lower edge cannot appear as an edge of a realistic roof. Hence, condition 1 is required.

Figure 6(a) shows the only possible configuration which violates condition 2 while satisfying condition 1. Consider two peak points of  $uv$  and  $u'v'$ ,  $s$  and  $s'$ . We cannot get a proper face between  $s$  and  $s'$  because of their heights. Therefore we cannot get a realistic roof which contains  $uv$  and  $u'v'$  in this case.

Suppose that  $uv$  and  $u'v'$  satisfy both conditions 1 and 2. First, divide  $P$  into three pieces,  $P_{1a}, P_{2a}$  and  $P_{3a}$  along  $uv$ . Next, divide one of  $P_{1a}, P_{2a}$  and  $P_{3a}$  which contains  $a'_1, a'_2$  and  $a'_3$ , into three pieces along  $u'v'$ . The latter division is possible because of condition 2. After two divisions, we get 5 rectilinear subpolygons of  $P$ , denoted by  $P_1, \dots, P_5$ . By taking the upper envelope of the roofs  $R^*(P_i)$ , we can get a realistic roof which contains  $uv$  and  $u'v'$ .  $\square$

**Lemma 8.** Let  $(a_1, a_2, a_3)$  be a candidate triple for a half-open valley  $uv$  and  $(a'_1, a'_2)$  be a candidate pair for an open valley  $u'v'$ .  $uv$  and  $u'v'$  are compatible if

1. Both  $a'_1$  and  $a'_2$  are contained in one of  $\partial P \setminus \{a_1, a_2, a_3\}$ .
2. Let  $P_{1a}, P_{2a}$ , and  $P_{3a}$  be rectilinear subpolygon of  $P$  divided by  $uv$ , as we did before. If condition 1 is hold, then  $a'_1$  and  $a'_2$  are contained in one of  $P_{1a}, P_{2a}$ , and  $P_{3a}$ , denoted by  $P_a$ . If  $P_{1a} = P_a$ , then  $r_{1a}$  of  $uv$ , denoted by  $r_{1a}$ , and the smallest axis parallel rectangle containing  $a$  and  $a'$ , denoted by  $B_{a'_1 a'_2}$ , satisfy the pseudo-disk property:  $r_{1a}$  and  $B_{a'_1 a'_2}$  do not cross.

*Proof.* Lemma 4 shows that a half-open valley must be connected to three reflex vertices via 5 reflex edges. Also an open valley must be connected to two reflex vertices via two reflex edges. As we see in the proof of Lemma 7, violating condition 1 makes some reflex edges of  $uv$  and  $u'v'$  crossing in their projection on the  $xy$ -plane, so condition 1 is required.

Suppose that  $r_{1a}$  and  $B_{a'_1 a'_2}$  violate the pseudo-disk property. From the height difference between the peak point of  $uv$  and the open valley  $u'v'$ , we can not get a proper face between them (Figure 6(b)). Therefore we cannot get a realistic roof which contains  $uv$  and  $u'v'$ .

Ahn et al. showed how to construct a realistic roof  $R$  over  $P$  with a candidate pair  $(a'_1, a'_2)$  for an open valley  $u'v'$ : Divide  $P$  into two pieces along  $\overline{C}_{a'_1 a'_2}$ ; Make two rectilinear subpolygon  $P'_1$  and  $P'_2$  by attaching  $B_{a'_1 a'_2}$  to each divided part; Take the upper envelope of  $R^*(P'_1) \cup R^*(P'_2)$ .

Suppose that  $uv$  and  $u'v'$  satisfy both condition 1 and 2. First, divide  $P$  into three pieces,  $P_{1a}, P_{2a}$ , and  $P_{3a}$ . Next, divide  $P_a$  which contains  $a'_1$  and  $a'_2$  into two pieces by using  $u'v'$ . After that, we can get 4 rectilinear subpolygons of  $P$ , denoted by  $P_1, \dots, P_4$ . By taking the upper envelope of the roofs  $R^*(P_i)$ , we can get a realistic roof which contains  $uv$  and  $u'v'$ .  $\square$

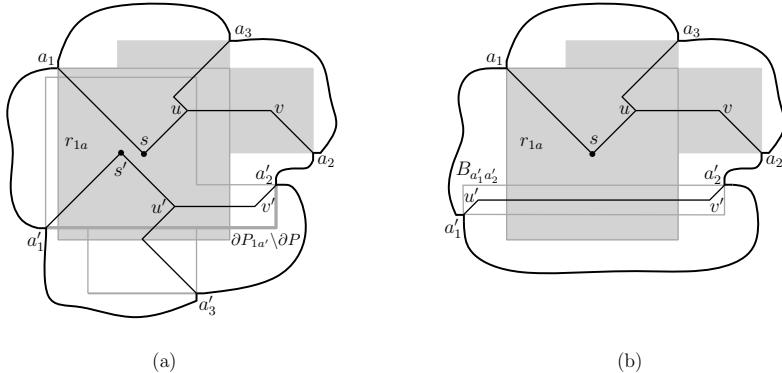
Let  $V$  be a set of candidate pairs and candidate triples. If any two elements of  $V$  satisfy Lemma 6 or Lemma 7 or Lemma 8, we can find a unique realistic roof  $R$  whose valleys correspond to  $V$ . Also, we call such  $V$  as a compatible set of  $P$ . Now we can conclude the following theorem.

**Theorem 1.** Let  $P$  be a rectilinear polygon with  $n$  vertices and  $V$  be a compatible set of  $k$  candidate pairs and  $l$  candidate triples with respect to  $P$ . Then there exists a unique realistic roof  $R$  whose valleys correspond to  $V$ . In addition, there exist  $k + 2l + 1$  rectilinear polygons  $P_1, \dots, P_{k+2l+1}$  contained in  $P$  such that

1.  $\cup_i P_i = P$ ,  $i = 1, \dots, k + 2l + 1$ .
2.  $R$  coincides with the upper envelope of  $R^*(P_i)$  for all  $i = 1, \dots, k + 2l + 1$ .

## 5 The Number of Realistic Roofs

We give an upper bound of the number of possible realistic roofs over  $P$  in terms of  $n$ . For this, we need a few technical lemmas.



**Fig. 6.** (a)  $r_{1a}$ ,  $r_1$  of  $uv$ , meets  $\partial P_{1a'}/\partial P$ . Between two peak points  $s$  and  $s'$ , we cannot construct a roof face. (b) Between the peak point  $s$  the open valley  $uv$ , we cannot construct a roof face.

**Lemma 9.** Let  $(a_1, a_2, a_3)$  be a candidate triple, where  $a_1$  and  $a_2$  have opposite orientations. Then  $(a_1, a_2)$  is also a candidate pair.

*Proof.* The candidate triple  $(a_1, a_2, a_3)$  admits a half open valley  $uv$ . The free space of  $uv$  contains  $B_{a_1 a_2}$ , so  $a_1$  and  $a_2$  admit an open valley  $u'v'$  related to  $uv$ .  $\square$

**Lemma 10.** Let  $(a_1, a_2, a_3)$  be a candidate triple for a half-open valley  $uv$ , where  $a_1$  and  $a_2$  have opposite orientations. If a candidate pair  $(a_4, a_5)$  is compatible with  $(a_1, a_2, a_3)$ , then  $(a_3, a_4, a_5)$  is not a candidate triple.

*Proof.* Without loss of generality, assume that the three reflex vertices  $a_1, a_2, a_3$  and the valley  $uv$  located as in Figure 7(a). By Lemma 8, both  $a_4$  and  $a_5$  must be contained one of three rectilinear subpolygons of  $P$ ,  $P_1, P_2$  and  $P_3$  defined by  $uv$ , as we did before. Assume to the contrary that  $(a_3, a_4, a_5)$  is a candidate triple for a half-open valley  $u'v'$ .

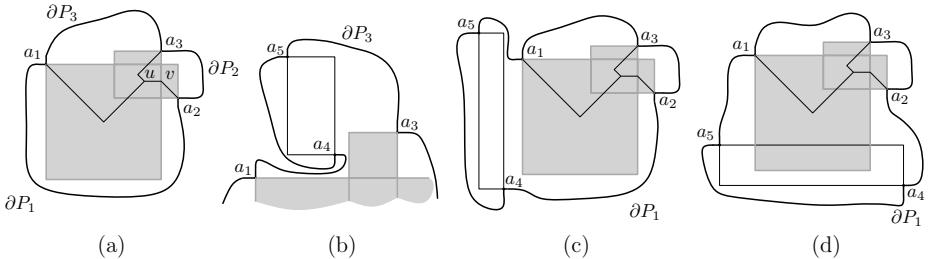
*Case 1.*  $a_4, a_5 \in \partial P_3$ . There is only one possible configuration (Figure 7(b)). By some careful case analysis, we have  $d_h(a_5, a_3) > d_h(a_4, a_3) > d_v(a_4, a_3)$ , which makes  $a_4$  be contained in the interior of the free space of  $u'v'$ .

*Case 2.*  $a_4, a_5 \in \partial P_2$ . There is no possible configuration.

*Case 3.*  $a_4, a_5 \in \partial P_1$ . There are two possible configurations. In case of Figure 7(c), we have  $d_h(a_5, a_3) > d_h(a_1, a_3) > d_v(a_1, a_3)$ , which makes  $a_1$  be contained in the interior of the free space of  $u'v'$ . In case of Figure 7(d), we have  $d_v(a_4, a_3) > d_h(a_1, a_3) > d_v(a_1, a_3)$ , which again makes  $a_1$  be contained in the interior of the free space of  $u'v'$ .  $\square$

Now we can find an upper bound of the number of realistic roofs over  $P$ .

**Theorem 2.** Let  $P$  be a rectilinear polygon with  $n$  vertices. There are at most  $1.3211^m \binom{m}{\lfloor \frac{m}{2} \rfloor}$  distinct realistic roofs over  $P$ , where  $m = \frac{n-4}{2}$ .



**Fig. 7.** Illustration of the proof of Lemma 10. Gray regions are free spaces.

*Proof.* Let  $R$  be a realistic roof over  $P$  with a half-open valley  $uv$ . By Lemma 9, we can get an open valley  $u'v'$  related to  $uv$ . Therefore, we can get a new realistic roof by replacing  $uv$  to  $u'v'$ . By repeating this process, we can get a realistic roof  $R'$  which does not contain any half-open valleys. It means that for any realistic roof  $R$  over  $P$ , there exist a unique realistic roof  $R'$  which has no half-open valleys. We can get the number of distinct realistic roofs over  $P$  with two steps: counting the number of realistic roofs  $R'$  over  $P$  which has no half-open valleys and counting the number of realistic roofs  $R$  which can be transformed to each  $R'$  by replacing its half-open valleys to related open valleys.

Ahn et al. [1,2] proved that the number of realistic roofs  $R'$  over  $P$  which has no half-open valleys is at most  $\binom{m}{\lfloor \frac{m}{2} \rfloor}$ , where  $m = \frac{n-4}{2}$ . We calculate the number of realistic roofs  $R$  over  $P$  corresponding to each  $R'$ . Suppose that  $R'$  contains  $k$  open valleys,  $u_1v_1, u_2v_2, \dots, u_kv_k$ .  $P$  has  $m - 2k$  reflex vertices that are not used to make open valleys. Let us call these reflex vertices as free vertices of  $R'$ . By Lemma 10, each of free vertices can make a half-open valley with at most one valley of  $u_1v_1, u_2v_2, \dots, u_kv_k$ . Let  $x_i$ ,  $1 \leq i \leq k$ , be the number of free vertices of  $R'$  which can make a half-open valley with  $u_iv_i$ . Then the number of realistic roofs that can be transformed to  $R'$  is at most  $(x_1+1)(x_2+1)\dots(x_k+1)$ , where  $x_1 + x_2 + \dots + x_k \leq m - 2k$ . From the inequality of arithmetic and geometric means, we can get  $(x_1+1)(x_2+1)\dots(x_k+1) \leq (\frac{x_1+x_2+\dots+x_k+k}{k})^k \leq (\frac{m-k}{k})^k = ((\frac{m}{k}-1)\frac{k}{m})^m$ . For a positive real number  $x$ ,  $\sup\{(x-1)^{\frac{1}{x}}\} \approx 1.3210997\dots$ , so  $((\frac{m}{k}-1)\frac{k}{m})^m < 1.3211^m$ . Therefore, we can get at most  $1.3211^m$  different realistic roofs over  $P$  corresponding to each  $R'$ , and the total number of distinct realistic roofs over  $P$  is at most  $1.3211^m \binom{m}{\lfloor \frac{m}{2} \rfloor}$ .  $\square$

## 6 Algorithm

In this section, we will present an algorithm that generates all possible realistic roofs over given rectilinear polygon  $P$ . Ahn et al. [1,2] suggested an efficient algorithm that generates all realistic roofs which do not have half-open valleys. Let us call the algorithm *Ahn's algorithm*. *Ahn's algorithm* uses  $O(n^4)$  time as preprocessing and generates realistic roofs one by one in  $O(1)$  time each.

We also use  $O(n^4)$  preprocessing time.  $P$  has  $O(n^3)$  triples and  $O(n^2)$  pairs of reflex vertices, and checking whether each triple and pair is a candidate triple or candidate pair takes  $O(n)$  time. After  $O(n^4)$  time, we can get all candidate triples and pairs of  $P$ . Create an empty list of reflex vertices for each candidate pair and add a reflex vertex  $a_i$  to a candidate pair's list if  $a_i$  and the candidate pair form a candidate triple.

Our algorithm works as follows. Run *Ahn's algorithm* and get a realistic roof  $R$  with  $k$  open valleys  $u_1v_1, \dots, u_kv_k$ . A candidate pair  $(a_i, a'_i)$  corresponding to  $u_iv_i$ ,  $1 \leq i \leq k$ , has a list of reflex vertices and let  $x_i$  be a reflex vertex chosen from the list. If we do not choose any vertex from the list of  $(a_i, a'_i)$ , let  $x_i = \emptyset$ . For the chosen vertices  $x_1, \dots, x_k$ , check whether the set of candidate pairs and triples  $V = \{(a_1, a'_1, x_1), \dots, (a_k, a'_k, x_k)\}$  is a compatible set of  $P$  where  $(a_i, a'_i, \emptyset) = (a_i, a'_i)$ . If  $(x_1, \dots, x_k) = (\emptyset, \dots, \emptyset)$ ,  $R$  is the realistic roof whose valleys correspond to  $V$ . By changing  $(x_1, \dots, x_k)$  one by one, checking the compatibility of  $V$  takes  $O(k)$  time. Suppose that  $(\dots, x_i, \dots)$  is changed to  $(\dots, x'_i, \dots)$ . We already know compatibilities between valleys which are induced by  $(a_j, a'_j, x_j)$  for  $j = 1, \dots, k$  and keep the total number of "conflicts" between the valleys. Check compatibilities between the valley induced by  $(a_i, a'_i, x_i)$  and the others, and decrease the total number of conflicts when it is not compatible with others. Next, check compatibilities between the valley induced by  $(a_i, a'_i, x'_i)$  and the others, and increase the total number of conflicts when it is not compatible with others. After that, if the total number of conflicts is zero, then the set  $V$  is compatible for  $P$ . Therefore, our algorithm finds all realistic roofs correspond to each  $R$  in  $O(m1.3211^m)$  time.

**Theorem 3.** *Given a rectilinear polygon  $P$  with  $n$  vertices,  $m$  of which are reflex vertices, after  $O(n^4)$ -time preprocessing, all the compatible sets of  $P$  can be enumerated in  $O(m1.3211^m(\lfloor \frac{m}{2} \rfloor))$ .*

## References

1. Ahn, H.-K., Bae, S.W., Knauer, C., Lee, M., Shin, C.-S., Vigneron, A.: Generating realistic roofs over a rectilinear polygon. In: Asano, T., Nakano, S.-I., Okamoto, Y., Watanabe, O. (eds.) ISAAC 2011. LNCS, vol. 7074, pp. 60–69. Springer, Heidelberg (2011)
2. Ahn, H.-K., Bae, S.W., Knauer, C., Lee, M., Shin, C.-S., Vigneron, A.: Realistic roofs over a rectilinear polygon. Submitted to a Journal. Personal Communication
3. Aichholzer, O., Albertsa, D., Aurenhammer, F., Gärtner, B.: A novel type of skeleton for polygons. J. Universal Comput. Sci. 1, 752–761 (1995)
4. Aichholzer, O., Aurenhammer, F.: Straight skeletons for general polygonal figures in the plane. In: Cai, J.-Y., Wong, C.K. (eds.) COCOON 1996. LNCS, vol. 1090, pp. 117–226. Springer, Heidelberg (1996)
5. Brenner, C.: Interactive modelling tools for 3D building reconstruction. In: Fritsch, D., Spiller, R. (eds.) Photogrammetric Week 1999, pp. 23–34 (1999)
6. Brenner, C.: Towards fully automatic generation of city models. Int. Archives of Photogrammetry and Remote Sensing XXXIII(pt. B3), 85–92 (2000)

7. Khoshelham, K., Li, Z.L.: A split-and-merge technique for automated reconstruction of roof planes. *Photogrammetric Engineering and Remote Sensing* 71(7), 855–863 (2005)
8. Krauß, T., Lehner, M., Reinartz, P.: Generation of coarse 3D models of urban areas from high resolution stereo satellite images. *Int. Archives of Photogrammetry and Remote Sensing* XXXVII, 1091–1098 (2008)
9. Laycock, R.G., Day, A.M.: Automatically generating large urban environments based on the footprint data of buildings. In: Proc. 8th ACM Sympos. Solid Model. Appl., pp. 346–351 (2003)
10. Sohn, G., Huang, X.F., Tao, V.: Using a binary space partitioning tree for reconstructing polyhedral building models from airborne lidar data. *Photogrammetric Engineering and Remote Sensing* 74(11), 1425–1440 (2008)

# Parametric Power Supply Networks

## (Extended Abstract)

Shiho Morishita and Takao Nishizeki

Department of Informatics, Faculty of Science and Technology  
Kwansei Gakuin University  
2-1 Gakuen, Sanda 669-1337, Japan  
[morishita0731@gmail.com](mailto:morishita0731@gmail.com), [nishi@kwansei.ac.jp](mailto:nishi@kwansei.ac.jp)

**Abstract.** Suppose that each vertex of a graph  $G$  is either a supply vertex or a demand vertex and is assigned a supply or a demand. All demands and supplies are nonnegative constant numbers in a steady network, while they are functions of a variable  $\lambda$  in a parametric network. Each demand vertex can receive “power” from exactly one supply vertex through edges in  $G$ . One thus wishes to partition  $G$  to connected components by deleting edges from  $G$  so that each component has exactly one supply vertex whose supply is at least the sum of demands in the component. The “partition problem” asks whether  $G$  has such a partition. If  $G$  has no such partition, one wishes to find the maximum number  $r^*$ ,  $0 \leq r^* < 1$ , such that  $G$  has such a partition when every demand is reduced to  $r^*$  times the original demand. The “maximum supply rate problem” asks to compute  $r^*$ . In this paper, we deal with a network in which  $G$  is a tree, and first give a polynomial-time algorithm for the maximum supply rate problem for a steady tree network, and then give an algorithm for the partition problem on a parametric tree network, which takes pseudo-polynomial time if all the supplies and demands are piecewise linear functions of  $\lambda$ .

## 1 Introduction

Consider a graph  $G$  in which each vertex is either a *supply vertex* or a *demand vertex* and is assigned a supply or a demand. Such a graph  $G$  is called a *power supply network*. All the supplies and demands are nonnegative constant numbers in an ordinary network, called a *steady network*, which has been considered so far [6,7,8,9,10]. This paper introduces a *parametric power supply network*, in which all the supplies and demands are functions of a parameter  $\lambda$ . The supply of a vertex  $v$  is denoted by  $s_v(\lambda)$  and the demand by  $d_v(\lambda)$ . Figure 1 depicts steady networks; each supply vertex is drawn by a square, each demand vertex by a circle, and the supply or demand is written inside. Figure 2(a) depicts a parametric network, whose variable demands  $d_{v_3}(\lambda)$  and  $d_{v_4}(\lambda)$  are drawn in Fig. 2(b).

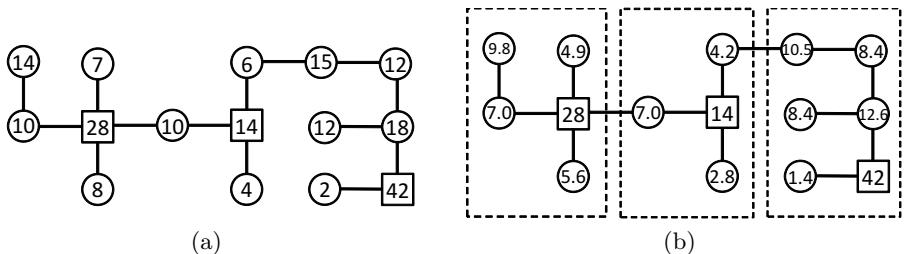
Each demand vertex  $v$  must receive an amount  $d_v(\lambda)$  of “power” or “commodity” from exactly one supply vertex through edges in a network  $G$ , while

each supply vertex  $v$  can supply, to demand vertices, at most an amount  $s_v(\lambda)$  of “power” in total. One thus wishes to partition  $G$  into connected components by deleting edges from  $G$  so that each component  $C$  has exactly one supply vertex whose supply is at least the sum of all demands in  $C$ . Such a partition is called a *feasible partition* of  $G$ . The *partition problem* asks, for each value of  $\lambda$ , whether  $G$  has a feasible partition. If  $G$  has no feasible partition for some value of  $\lambda$ , one wishes to find the maximum number  $r^*$ ,  $0 \leq r^* < 1$ , such that  $G$  has a feasible partition for the value if every demand  $d_v(\lambda)$  is uniformly reduced to a new demand  $d'_v(\lambda) = r^* \cdot d_v(\lambda)$ . We call  $r^*$  the *maximum supply rate*, and call the problem of computing  $r^*$  the *maximum supply rate problem*. The maximum supply rate problem for a steady network is a special case of the partition problem for a parametric network, as will be observed in Section 2.

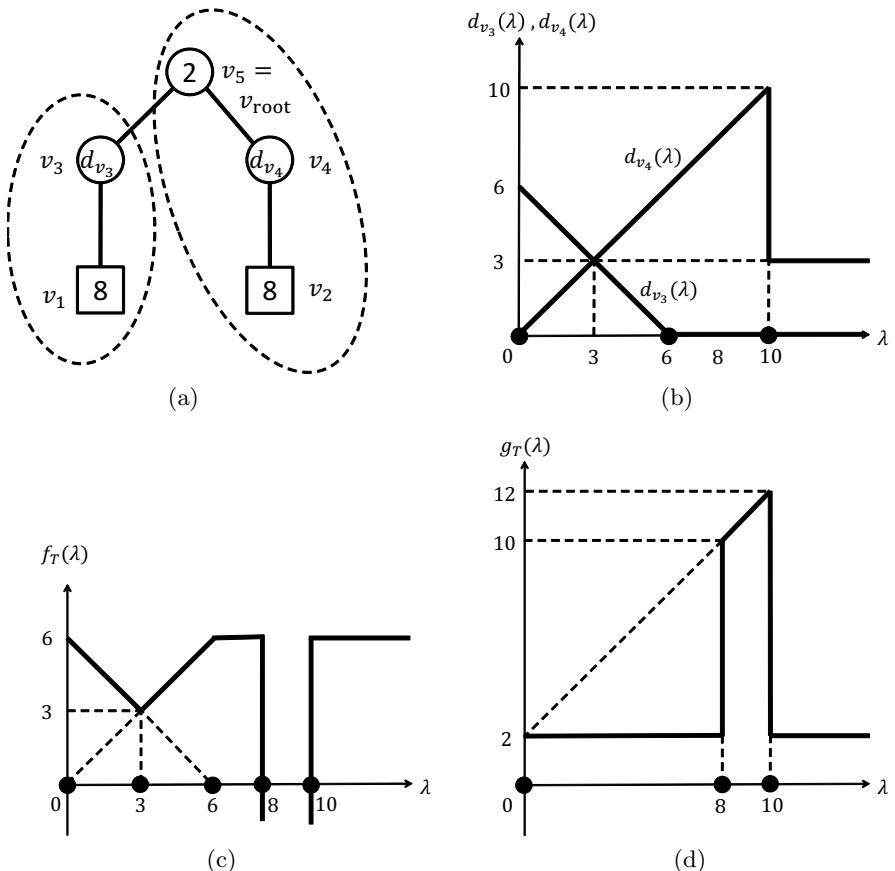
The steady network in Fig. 1(a) has no feasible partition, and the maximum supply rate  $r^*$  is 0.7; Fig. 1(b) depicts a new network with  $d'_v = 0.7 \cdot d_v$  and illustrates a feasible partition by dotted lines.

The partition problem and the maximum supply rate problem have some applications to the power supply problem for power delivery networks, in which a supply or demand may depend on a parameter  $\lambda$  such as time, temperature, oil price, etc.[1,11,12,13]. The partition problem is NP-complete even for a steady network on a series-parallel graph, because the “set partition problem” [5, p. 47] can be easily reduced to the partition problem for a steady network on a complete bipartite graph  $K_{2,n-2}$ , which is a series-parallel graph [9]. Therefore, the maximum supply rate problem is NP-hard even for series-parallel steady networks. Hence, it is very unlikely that these problems can be solved in polynomial time even for series-parallel steady networks. However, the partition problem can be solved for steady tree networks in linear time [9].

In this paper, we first give a polynomial-time algorithm to solve the maximum supply rate problem for a steady tree network  $T$ . It takes time  $O(nL)$ , where  $n$  is the number of vertices in  $T$  and  $L$  is the logarithmic size of  $T$ . We then present an algorithm to solve the partition problem for a parametric tree network. It takes pseudo-polynomial time if all supplies and demands are piecewise linear functions with integer coefficients. More precisely, it takes time  $O(nW^2)$ , where  $W$  is the sum of absolute values of all integer coefficients of supplies and demands.



**Fig. 1.** (a) Steady tree network  $T$  with no feasible partition, and (b) new network constructed from  $T$  for the maximum supply rate  $r^* = 0.7$



**Fig. 2.** (a) Parametric tree network  $T$  rooted at  $v_5$ , (b) variable demands  $d_{v_3}(\lambda)$  and  $d_{v_4}(\lambda)$ , (c) surplus  $f_T(\lambda)$ , and (d) deficit  $g_T(\lambda)$  of  $T$

## 2 Maximum Supply Rate Problem

In this section we deal with steady networks in which all supplies and demands are positive integers, and show that the maximum supply rate problem can be solved for steady tree networks in polynomial time.

Let  $G = (V, E)$  be a steady network, where  $V$  is the set of vertices and  $E$  is the set of edges of  $G$ . Let  $V_s$  be the set of all supply vertices, and let  $V_d$  be the set of all demand vertices, then  $V = V_s \cup V_d$  and  $V_s \cap V_d = \emptyset$ . Let  $n = |V|$  and  $n_s = |V_s|$ . We denote by  $d_v$  the positive integral demand of a demand vertex  $v \in V_d$ , and by  $s_v$  the positive integral supply of a supply vertex  $v \in V_s$ . The *partition problem* asks whether  $V$  can be partitioned to a number  $n_s$  of subsets  $V_1, V_2, \dots, V_{n_s}$  such that each  $V_i$ ,  $1 \leq i \leq n_s$ , induces a subtree of  $T$ , contains exactly one supply vertex, say  $u$ , and

$$\sum_{v \in V_i / \{u\}} d_v \leq s_u.$$

We call such a partition of  $V$  a *feasible partition of  $G$* . Ito *et al.* [9] gave an algorithm to solve the partition problem in time  $O(n)$  if  $G$  is a tree. We call the algorithm.

**Partition.** Note that our parametric algorithm in Section 3 also runs in time  $O(n)$  for a steady tree network.

The *maximum supply rate problem* for  $G$  asks to find the maximum number  $r (> 0)$  such that  $G$  has a feasible partition if the demand  $d_v$  is replaced by a new demand  $d'_v = r \cdot d_v$  for every demand vertex  $v$ . We call the maximum value  $r^*$  of such a number  $r$  the *maximum supply rate of  $G$* . Thus the maximum supply rate  $r^*$  may be greater than 1. When  $r^* < 1$ , the value  $1 - r^*$  is called the *minimum power saving rate of  $G$* .

The maximum supply rate problem for a steady network  $G$  can be formulated as a partition problem for a parametric network  $G_{\text{para}}$ , in which the demand of every demand vertex  $v$  is a linear function  $d_v(\lambda) = d_v \cdot \lambda$  and the supply of every supply vertex  $u$  is a constant function  $s_u(\lambda) = s_u$ . The maximum supply rate  $r^*$  of  $G$  is equal to the maximum value of  $\lambda$  for which  $G_{\text{para}}$  has a feasible partition.

Obviously, the following lemma holds.

**Lemma 1.** Suppose that a steady network  $G$  has a feasible partition when every demand  $d_v$  is replaced by  $d'_v = r \cdot d_v$  for a positive real number  $r$ . Then, for any number  $r'$  with  $0 \leq r' \leq r$ ,  $G$  has a feasible partition when every demand  $d_v$  is replaced by  $d'_v = r' \cdot d_v$ .

One can thus compute  $r^*$  for a steady tree network  $T$  by a binary search on the infinite set of all positive real numbers  $r$  with the aid of the algorithm **Partition**. However, such a simple binary search either cannot exactly compute  $r^*$  or does not run in polynomial time. The idea of our algorithm is to notice that  $r^*$  is a rational number, as follows.

**Lemma 2.** Let  $r^*$  be the maximum supply rate of a steady network  $G = (V, E)$ , and let  $S = \prod_{v \in V_s} s_v$  and  $D = \sum_{v \in V_d} d_v$ . Then

$$r^* \in \{S/z \mid z \text{ is an integer and } S \cdot D \geq z \geq 1\}.$$

*Proof.* Since  $r^*$  is the maximum supply rate, there is a partition of  $V$  to subsets  $V_1, V_2, \dots, V_{n_s}$  such that each  $V_i$ ,  $1 \leq i \leq n_s$ , contains exactly one supply vertex, say  $u$ , and

$$\sum_{v \in V_i / \{u\}} r^* \cdot d_v \leq s_u.$$

The inequality above holds in equality for some  $i$ ,  $1 \leq i \leq n_s$ , that is,

$$r^* \sum_{v \in V_i / \{u\}} d_v = s_u;$$

otherwise,  $r^*$  would not be the maximum supply rate. Let  $z^* = S/r^*$ , then from the two equations above we have

$$z^* = \left( \frac{\prod_{v \in V_s} s_v}{s_u} \right) \cdot \sum_{v \in V_i / \{u\}} d_v.$$

Thus  $z^*$  is an integer, and  $1 \leq z^* \leq S \cdot D$ . Therefore,  $r^*$  ( $= S/z^*$ ) is equal to  $S/z$  for some integer  $z$ ,  $1 \leq z \leq S \cdot D$ .  $\square$

Thus, one can find the maximum supply rate  $r^*$  of a steady tree network  $T = (V, E)$  by a binary search on the finite set  $\{S/z \mid S \cdot D \geq z \geq 1\}$  of rational numbers with the aid of **Partition** in time  $O(n \log_2(S \cdot D))$ .

The *logarithmic size*  $L$  of a steady network  $T$  is

$$L = \sum_{u \in V_s} \lceil \log_2(s_v + 1) \rceil + \sum_{v \in V_d} \lceil \log_2(d_v + 1) \rceil,$$

and clearly  $\log_2(S \cdot D) \leq L$ . We thus have the following theorem.

**Theorem 1.** The maximum supply rate problem can be solved in time  $O(nL)$  for a steady tree network  $T$ , where  $n$  is the number of vertices and  $L$  is the logarithmic size of  $T$ .

### 3 Parametric Networks

In this section we present an algorithm to solve the partition problem for a parametric tree network  $T$ .

#### 3.1 Definitions

One may assume without loss of generality that a tree  $T$  is rooted at an arbitrarily chosen vertex  $v_{\text{root}}$ . We also assume that all supplies  $s_v(\lambda)$  and demands  $d_v(\lambda)$  in  $T$  are functions of a common nonnegative real variable  $\lambda (\geq 0)$ .

A *feasible partition*  $\pi_\lambda = (V_1, V_2, \dots, V_{n_s})$  of a rooted tree network  $T = (V, E)$  for a value  $\lambda$  is a partition of  $V$  to a number  $n_s$  of subsets  $V_1, V_2, \dots, V_{n_s}$  such that

- (a) the root of  $T$  is contained in  $V_1$ , that is,  $v_{\text{root}} \in V_1$ ; and
- (b) each  $V_i$ ,  $1 \leq i \leq n_s$ , induces a subtree of  $T$ , contains exactly one supply vertex, say  $u$ , and

$$\sum_{v \in V_i / \{u\}} d_v(\lambda) \leq s_u(\lambda).$$

The partition problem asks to find every value of  $\lambda$  for which  $T$  has a feasible partition. We actually find every interval of nonnegative real numbers such that  $T$  has a feasible partition  $\pi_\lambda$  for each value  $\lambda$  in the interval.

For the network  $T$  in Fig. 2(a),  $v_5 = v_{\text{root}}$ ,  $s_{v_1}(\lambda) = s_{v_2}(\lambda) = 8$ ,  $d_{v_5}(\lambda) = 2$ , and the variable demands  $d_{v_3}(\lambda)$  and  $d_{v_4}(\lambda)$  are drawn in Fig. 2(b). A feasible partition of  $T$  for  $0 \leq \lambda \leq 6$  is indicated by dotted lines in Fig. 2(a). The solution for  $T$  is a set of two intervals  $[0, 8]$  and  $[10, \infty)$ .

We find a feasible partition of  $T$  by the bottom-up computation on a rooted tree  $T$ . More precisely, we find a feasible partition and an extended partition, called a “root-feasible partition,” from those of smaller subtrees.

A *root-feasible partition*  $\pi_\lambda = (V_1, V_2, \dots, V_{n_s+1})$  of  $T$  for  $\lambda$  is a partition of  $V$  to a number  $n_s + 1$  of subsets  $V_1, V_2, \dots, V_{n_s+1}$  such that

- (a)  $v_{\text{root}} \in V_1$  and  $V_1 \cap V_s = \emptyset$ ; and
- (b) each  $V_i$ ,  $2 \leq i \leq n_s + 1$ , induces a subtree of  $T$ , contains exactly one supply vertex, say  $u$ , and

$$\sum_{v \in V_i / \{u\}} d_v(\lambda) \leq s_u(\lambda).$$

Thus,  $T$  has no root-feasible partition for any  $\lambda$  if  $v_{\text{root}}$  is a supply vertex. (A root-feasible partition of  $T$  in Fig. 2(a) for  $0 \leq \lambda \leq 8$  is  $(\{v_5\}, \{v_1, v_3\}, \{v_2, v_4\})$ .)

Let  $\pi_\lambda = (V_1, V_2, \dots, V_{n_s})$  be a feasible partition of  $T$  for a value  $\lambda$ , and let  $u$  be the supply vertex in  $V_1$ . Then the *surplus*  $\text{surp}(\pi_\lambda)$  of  $\pi_\lambda$  is

$$\text{surp}(\pi_\lambda) = s_u(\lambda) - \sum_{v \in V_1 / \{u\}} d_v(\lambda).$$

We now define a function  $f_T(\lambda)$ , called the *surplus of a parametric tree network*  $T$ , as follows:

$$f_T(\lambda) = \max_{\pi_\lambda} \text{surp}(\pi_\lambda)$$

where the maximum is taken over all feasible partitions  $\pi_\lambda$  of  $T$  for  $\lambda$ . Let  $f_T(\lambda) = -\infty$  if  $T$  has no feasible partition for  $\lambda$ . Intuitively,  $f_T(\lambda)$  is the maximum amount of power that can be delivered outside  $T$  through the root when all demand vertices are supplied power. (Figure 2(c) depicts  $f_T(\lambda)$  for  $T$  in Fig. 2(a).)

Let  $\pi_\lambda = (V_1, V_2, \dots, V_{n_s+1})$  be a root-feasible partition of  $T$  for a value  $\lambda$ . Then  $v_{\text{root}} \in V_1$ ,  $v_{\text{root}}$  is a demand vertex, and  $V_1$  contains no supply vertex. The *deficit*  $\text{def}(\pi_\lambda)$  of  $\pi_\lambda$  is

$$\text{def}(\pi_\lambda) = \sum_{v \in V_1} d_v(\lambda).$$

We now define a function  $g_T(\lambda)$ , called the *deficit of  $T$* , as follows:

$$g_T(\lambda) = \min_{\pi_\lambda} \text{def}(\pi_\lambda)$$

where the minimum is taken over all root-feasible partitions  $\pi_\lambda$  of  $T$  for  $\lambda$ . Let  $g_T(\lambda) = +\infty$  if  $T$  has no root-feasible partition for  $\lambda$ . Thus  $g_T(\lambda) = +\infty$  for any  $\lambda$  if  $v_{\text{root}}$  is a supply vertex. Intuitively,  $g_T(\lambda)$  is the minimum amount of power that must be delivered inside  $T$  through  $v_{\text{root}}$  when  $v_{\text{root}}$  and possibly some other demand vertices are supplied power from outside. (Figure 2(d) depicts  $g_T(\lambda)$  for  $T$  in Fig. 2(a).)

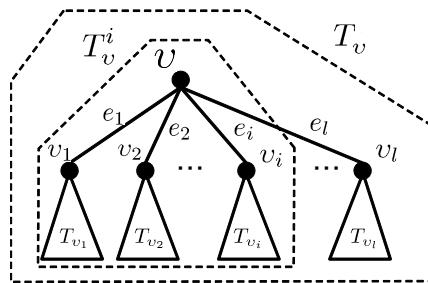
We similarly define the surplus  $f_{T'}(\lambda)$  and deficit  $g_{T'}(\lambda)$  for a rooted subtree  $T'$  of  $T$ .

For a vertex  $v$  of  $T$ , we denote by  $T_v$  the maximum subtree of  $T$  rooted at  $v$ . Let  $v_1, v_2, \dots, v_l$  be the children of  $v$  in  $T$ , and let  $e_i, 1 \leq i \leq l$ , be the edge joining  $v$  and  $v_i$ . Let  $T_{v_i}, 1 \leq i \leq l$ , be the maximum subtree of  $T$  rooted at  $v_i$ . We denote by  $T_v^i$  the subtree of  $T_v$  which consists of vertex  $v$ , edges  $e_1, e_2, \dots, e_i$  and subtrees  $T_{v_1}, T_{v_2}, \dots, T_{v_i}$ . In Fig. 3  $T_v$  and  $T_v^i$  are surrounded by dotted lines. Clearly  $T = T_{v_{\text{root}}}$  and  $T_v = T_v^l$ . We denote by  $T_v^0$  the subtree consisting of a single vertex  $v$ .

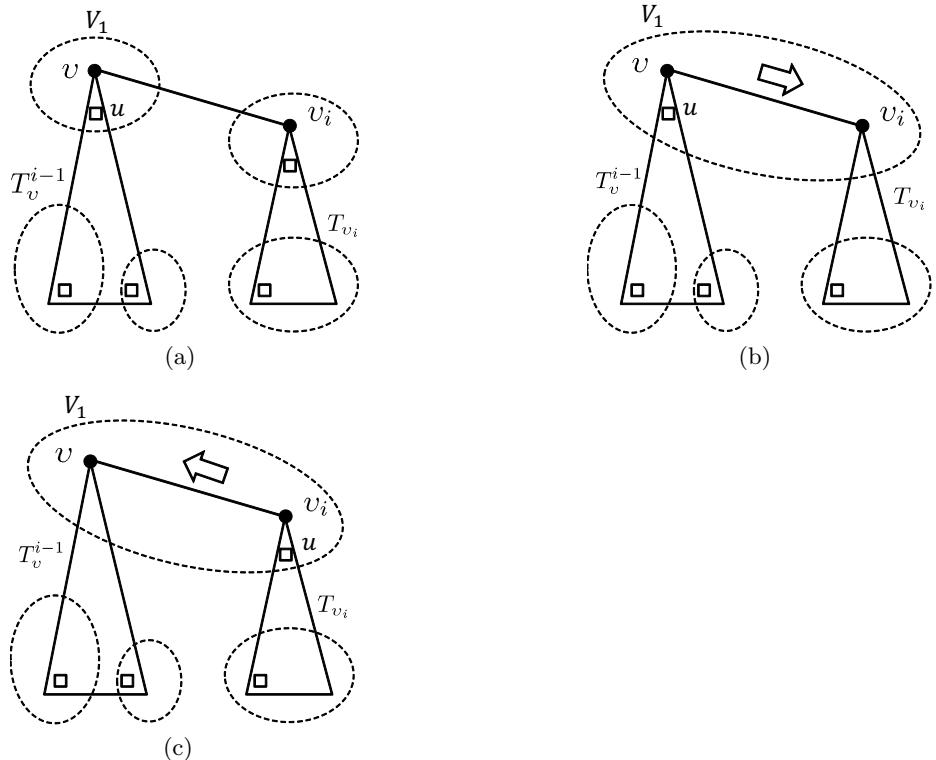
### 3.2 Algorithm

Our algorithm computes the surplus  $f_{T_v}(\lambda)$  and deficit  $g_{T_v}(\lambda)$  for each vertex  $v$  of  $T$  from leaves to the root of  $T$  by means of a dynamic programming approach, as described in (i)-(iii) below. From the surplus  $f_T(\lambda)$  of  $T = T_{v_{\text{root}}}$ , one can easily find every interval of nonnegative real numbers such that  $f_T(\lambda) \geq 0$  for every value  $\lambda$  in the interval. We output the set of all these intervals as the solution of the partition problem of a parametric tree network  $T$ .

(i) We first compute the surplus and deficit of  $T_v^0$  for each vertex  $v$  of  $T$  as follows. (Remember that  $T_v^0$  consists of a single vertex  $v$ .) If  $v$  is a supply vertex, then  $f_{T_v^0}(\lambda) = s_v(\lambda)$  and  $g_{T_v^0}(\lambda) = +\infty$  for every  $\lambda$ . If  $v$  is a demand vertex, then  $f_{T_v^0}(\lambda) = -\infty$  and  $g_{T_v^0}(\lambda) = d_v(\lambda)$  for every  $\lambda$ . Since  $T_v = T_v^0$  for every leaf  $v$  of  $T$ , we have thus computed  $f_{T_v}$  and  $g_{T_v}$  for every leaf  $v$  of  $T$ .



**Fig. 3.** Rooted subtrees



**Fig. 4.** Feasible partitions  $\pi_\lambda$  of  $T_v^i$

(ii) We next compute the surplus and deficit of a tree  $T_v^i$ ,  $1 \leq i \leq l$ , for each internal vertex  $v$  of  $T$  from those of its subtrees  $T_v^{i-1}$  and  $T_{v_i}$ , where  $l$  is the number of the children of  $T_v$ . Note that  $T_v = T_v^l$  and that  $T_v^i$  is obtained from  $T_v^{i-1}$  and  $T_{v_i}$  by joining  $v$  and  $v_i$  as illustrated in Fig. 4.

We now explain how to compute the surplus  $f_{T_v^i}$  of  $T_v^i$ . Let  $n_i$  be the number of supply vertices in  $T_v^i$ . Assume that  $f_{T_v^i}(\lambda) \neq -\infty$ , that is,  $T_v^i$  has a feasible partition for  $\lambda$ . Let  $\pi_\lambda = (V_1, V_2, \dots, V_{n_i})$  be a feasible partition of  $T_v^i$  such that  $f_{T_v^i}(\lambda) = \text{surp}(\pi_\lambda)$ . Then  $V_1$  contains the root  $v$  of  $T_v^i$ , as illustrated in Fig. 4 where  $\pi_\lambda$  is indicated by dotted lines and a supply vertex is drawn by a square. There are the following three cases to consider.

Case (a):  $v_i \notin V_1$ .

In this case,  $f_{T_{v_i}}(\lambda) \geq 0$ , and  $\pi_\lambda$  induces feasible partitions of  $T_v^{i-1}$  and  $T_{v_i}$ . (See Fig. 4(a).) For this case we compute the following function  $f_{T_v^i}^a$  from  $f_{T_v^{i-1}}$  and  $f_{T_{v_i}}$ :

$$f_{T_v^i}^a(\lambda) = \begin{cases} f_{T_v^{i-1}}(\lambda) & \text{if } f_{T_{v_i}}(\lambda) \geq 0; \\ -\infty & \text{otherwise.} \end{cases}$$

Case (b):  $v_i \in V_1$ , and the supply vertex  $u$  in  $V_1$  is contained in  $T_v^{i-1}$ .

In this case,  $f_{T_v^{i-1}}(\lambda) \geq g_{T_{v_i}}(\lambda)$ , and  $\pi_\lambda$  induces a feasible partition of  $T_v^{i-1}$  and a root-feasible partition of  $T_{v_i}$ . (In Fig. 4(b) the arrow attached to edge  $(v, v_i)$  indicates the direction of power flow through it.) For this case we compute the following function  $f_{T_v^i}^b$ :

$$f_{T_v^i}^b(\lambda) = \begin{cases} f_{T_v^{i-1}}(\lambda) - g_{T_{v_i}}(\lambda) & \text{if } f_{T_v^{i-1}}(\lambda) \geq g_{T_{v_i}}(\lambda); \\ -\infty & \text{otherwise.} \end{cases}$$

Case (c):  $v_i \in V_1$ , and the supply vertex  $u$  in  $V_1$  is contained in  $T_{v_i}$ .

In this case,  $g_{T_v^{i-1}}(\lambda) \leq f_{T_{v_i}}(\lambda)$ , and  $\pi_\lambda$  induces a root-feasible partition of  $T_v^{i-1}$  and a feasible partition of  $T_{v_i}$ . (See Fig. 4(c).) For this case we compute the following function  $f_{T_v^i}^c$ :

$$f_{T_v^i}^c(\lambda) = \begin{cases} -g_{T_v^{i-1}}(\lambda) + f_{T_{v_i}}(\lambda) & \text{if } g_{T_v^{i-1}}(\lambda) \leq f_{T_{v_i}}(\lambda); \\ -\infty & \text{otherwise.} \end{cases}$$

From the three functions  $f_{T_v^i}^a$ ,  $f_{T_v^i}^b$  and  $f_{T_v^i}^c$  above, we can now compute  $f_{T_v^i}$  as follows:

$$f_{T_v^i}(\lambda) = \max\{f_{T_v^i}^a(\lambda), f_{T_v^i}^b(\lambda), f_{T_v^i}^c(\lambda)\}.$$

One can similarly compute the deficit  $g_{T_v^i}$  of  $T_v^i$ . The detail is omitted in this extended abstract.

(iii) Repeating the computation in (ii) above for each edge  $(v, v_i)$  of  $T$ , we can compute  $f_T(\lambda)$  and  $g_T(\lambda)$ .

### 3.3 Computation Time

In this subsection we assume that all the supplies and demands are piecewise linear functions of a common variable  $\lambda (\geq 0)$ , and analyze the computation time of our algorithm.

A *breakpoint* of a piecewise linear function  $f$  is defined to be a point  $\lambda$  at which the slope of the curve of  $f$  changes, and the number of breakpoints of  $f$  is denoted by  $p(f)$ . For the sake of convenience, we assume that  $\lambda = 0$  is a breakpoint of  $f$ . (For example,  $p(d_{v_3}(\lambda)) = 2$  and  $p(f_T(\lambda)) = 5$  for  $d_{v_3}(\lambda)$  and  $f_T(\lambda)$  in Fig. 2 where a breakpoint is drawn as a black dot.) Then the *size*  $N$  of a parametric tree network  $T = (V, E)$  is

$$N = \sum_{v \in V_s} p(s_v(\lambda)) + \sum_{v \in V_d} p(d_v(\lambda)).$$

We define  $P$  as follows:

$$P = \max_{T'} \max\{p(f_{T'}(\lambda)), p(g_{T'}(\lambda))\}$$

where  $T'$  runs over all rooted subtrees of  $T$ . Note that  $f_{T'}$  and  $g_{T'}$  are piecewise linear functions.

Clearly one can compute the surplus  $f_{T_v^0}(\lambda)$  and deficit  $g_{T_v^0}(\lambda)$  of all vertices  $v$  in  $T$  in time  $O(N)$  as in (i) of Sect. 3.2.

One can compute  $f_{T_v^i}$  and  $g_{T_v^i}$  for tree  $T_v^i$  from those for its subtrees  $T_v^{i-1}$  and  $T_{v_i}$  in time  $O(P)$  as in (ii) of Sect. 3.2. (Note that the maximum and minimum of two piecewise linear functions can be computed by finding the upper and lower envelopes of the two piecewise linear curves, respectively.) The computation of (ii) occurs  $n - 1$  times since  $T$  has  $n - 1$  edges. Hence, one can compute  $f_T(\lambda)$  and  $g_T(\lambda)$  in time  $O(nP)$ . From  $f_T(\lambda)$  one can find, in time  $O(P)$ , all intervals such that  $T$  has a feasible partition  $\pi_\lambda$  for any value  $\lambda$  in each integral. Thus the partition problem can be solved in time  $O(nP)$ .

$P$  may be greater than  $N$ . (For example, neither the breakpoint  $\lambda = 3$  nor  $\lambda = 8$  of  $f_T(\lambda)$  is a breakpoint of any supply or demand of  $T$  in Fig. 2(a).) However,  $P$  is often bounded by a polynomial in  $N$  in many practical applications. In particular, if  $T$  is a steady network then  $P = 1$  and hence our algorithm takes time  $O(n)$ . If all supplies and demands are staircase functions, then  $P \leq N$  and hence our algorithm takes time  $O(nN)$ .

### 3.4 Bounds on $P$

In this subsection we assume that all the supplies and demands of  $T$  are piecewise linear functions with integer coefficients.

Let  $B$  be the number of breakpoints of supplies and demands, and let  $p_1, p_2, \dots, p_B$  be these points. (For  $T$  in Fig. 2(a)  $B = 3$  as indicated by three black dots in Fig. 2(b).) One may assume that  $0 = p_1 < p_2 < \dots < p_B$ , and let  $p_{B+1} = \infty$ . We now assume that

- (a) if  $v$  is a supply vertex and  $p_i < \lambda < p_{i+1}$ ,  $1 \leq i \leq B$ , then

$$s_v(\lambda) = a_{vi}\lambda + b_{vi}$$

for some integers  $a_{vi}$  and  $b_{vi}$  (possibly after multiplying them by the least common multiple of denominators); and

- (b) if  $v$  is a demand vertex and  $p_i < \lambda < p_{i+1}$ ,  $1 \leq i \leq B$ , then

$$d_v(\lambda) = a_{vi}\lambda + b_{vi}$$

for some integers  $a_{vi}$  and  $b_{vi}$ .

Let

$$W = \sum_{i=1}^B \sum_{v \in V} (|a_{vi}| + |b_{vi}|).$$

Then we show that  $P$  is bounded by a pseudo-polynomial, that is,  $P$  is bounded by a polynomial in  $W$ . More precisely, we have the following lemma, whose proof is omitted in this extended abstract.

**Lemma 3.**  $P = O(W^2)$ .

From the lemma above we have the following theorem.

**Theorem 2.** The partition problem for a parametric tree network can be solved in time  $O(nW^2)$  if all supplies and demands are piecewise linear functions with integer coefficients, where  $W$  is the sum of absolute values of all coefficients of supplies and demands.

Thus, our algorithm runs in polynomial time if  $W$  is polynomial in  $N$ .

## 4 Conclusions

In the paper we first showed that the maximum supply rate problem for a steady tree network  $T$  can be solved in time  $O(nL)$ , where  $n$  is the number of vertices in  $T$  and  $L$  is the logarithmic size of  $T$ . It would be interesting to obtain a strongly polynomial-time algorithm for the problem, whose computation time is bounded by a polynomial only in  $n$ .

We then gave an algorithm to solve the partition problem for a parametric tree network. The algorithm runs in pseudo-polynomial time if all the supplies and demands are piecewise linear functions with integer coefficients. We assumed for the sake of convenience that the supplies and demands are functions of a single variable  $\lambda$ . However, our algorithm in Section 3.2 can be easily extended to the case where the supplies and demands are functions of two or more variables.

Kawabata and Nishizeki [10] considered a steady tree network in which each edge is also assigned a constant edge-capacity, and gave a linear algorithm to solve the partition problem. Our algorithm for the maximum supply rate problem in Section 2 can be easily extended to the case of a steady tree network with constant edge-capacity, and our algorithm for the partition problem in Section 3.2 can be extended to the case of a parametric tree network in which edge capacity is also a function of  $\lambda$ . Note that our problems with (constant or functional) edge-capacity cannot be formulated by the multi-source multi-sink parametric flow problem or the unsplittable parametric flow problem [2,3,4,11].

If a tree network  $T$  has no feasible partition, one would like to solve the *maximum partition problem*, which asks to find a partition of  $T$  to subtrees such that

- (a) each subtree contains at most one supply vertex;
- (b) if a subtree contains a supply vertex, then the supply is no less than the sum of demands in the subtrees; and
- (c) the sum of demands in all subtrees, each containing a supply vertex, is maximum among all these partitions of  $T$ .

There are fully polynomial-time approximation schemes (FPTAS) for the problem on a steady tree network without edge-capacity [8] and on a steady tree network with constant edge-capacity [10]. It would be interesting to obtain an FPTAS for the problem on a parametric tree network with or without edge-capacity.

**Acknowledgments.** This research was supported by MEXT-Supported Program for the Strategic Research Foundation at Private Universities.

## References

1. Boulaxis, N.G., Papadopoulos, M.P.: Optimal feeder routing in distribution system planning using dynamic programming technique and GIS facilities. *IEEE Trans. Power Delivery* 17(1), 242–247 (2002)
2. Chekuri, C., Ene, A., Korula, N.: Unsplittable flow in paths and trees and column-restricted packing integer programs. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) APPROX and RANDOM 2009. LNCS, vol. 5687, pp. 42–55. Springer, Heidelberg (2009)
3. Chekuri, C., Mydlarz, M., Shepherd, F.B.: Multicommodity demand flow in a tree. *ACM Trans. on Algorithms* 3, Article 3 (2007)
4. Gallo, G., Grigoriadis, M.D., Tarjan, R.E.: A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.* 18(1), 30–55 (1989)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco (1979)
6. Ito, T., Demaine, E.D., Zhou, X., Nishizeki, T.: Approximability of partitioning graphs with supply and demand. *Journal of Discrete Algorithms* 6, 627–650 (2008)
7. Ito, T., Hara, T., Zhou, X., Nishizeki, T.: Minimum cost partitions of trees with supply and demand. *Algorithmica* 64, 400–415 (2012)
8. Ito, T., Zhou, X., Nishizeki, T.: Partitioning graphs of supply and demand. *Discrete Applied Math.* 157, 2620–2633 (2009)
9. Ito, T., Zhou, X., Nishizeki, T.: Partitioning trees of supply and demand. *Int. J. Found. Comput. Sci.* 16, 803–827 (2005)
10. Kawabata, M., Nishizeki, T.: Partitioning trees with supply, demand and edge-capacity. In: Proc. of ISORA 2011. Lecture Notes in Operation Research, vol. 14, pp. 51–58 (2011); also IEICE Trans. on Fundamentals of Electronics, Communications and Computer Science (to appear)
11. Minieka, E.: Parametric network flows. *Operation Research* 20(6), 1162–1170 (1972)
12. Morton, A.B., Mareels, I.M.Y.: An efficient brute-force solution to the network reconfiguration problem. *IEEE Trans. Power Delivery* 15, 996–1000 (2000)
13. Teng, J.-H., Lu, C.-N.: Feeder-switch relocation for customer interruption cost minimization. *IEEE Trans. Power Delivery* 17, 254–259 (2002)

# Approximating the Minimum Independent Dominating Set in Perturbed Graphs

Weitian Tong, Randy Goebel, and Guohui Lin<sup>\*</sup>

Department of Computing Science

University of Alberta

Edmonton, Alberta T6G 2E8, Canada

{weitian,rgoebel,guohui}@ualberta.ca

**Abstract.** We investigate the minimum independent dominating set in perturbed graphs  $\mathfrak{g}(G, p)$  of input graph  $G = (V, E)$ , obtained by negating the existence of edges independently with a probability  $p > 0$ . The minimum independent dominating set (MIDS) problem does not admit a polynomial running time approximation algorithm with worst-case performance ratio of  $n^{1-\epsilon}$  for any  $\epsilon > 0$ . We prove that the size of the minimum independent dominating set in  $\mathfrak{g}(G, p)$ , denoted as  $i(\mathfrak{g}(G, p))$ , is asymptotically almost surely in  $\Theta(\log |V|)$ . Furthermore, we show that the probability of  $i(\mathfrak{g}(G, p)) \geq \sqrt{\frac{4|V|}{p}}$  is no more than  $2^{-|V|}$ , and present a simple greedy algorithm of proven worst-case performance ratio  $\sqrt{\frac{4|V|}{p}}$  and with polynomial expected running time.

**Keywords:** Independent set, independent dominating set, dominating set, approximation algorithm, perturbed graph, smooth analysis.

## 1 Introduction

An *independent set* in a graph  $G = (V, E)$  is a subset of vertices that are pair-wise non-adjacent to each other. The independence number of  $G$ , denoted by  $\alpha(G)$ , is the size of a maximum independent set in  $G$ . One close notion to independent set is the *dominating set*, which refers to a subset of vertices such that every vertex of the graph is either in the subset or is adjacent to some vertex in the subset. In fact, an independent set becomes a dominating set if and only if it is maximal. The size of a minimum independent dominating set of  $G$  is denoted by  $i(G)$ , while the domination number of  $G$ , or the size of a minimum dominating set of  $G$ , is denoted by  $\gamma(G)$ . It follows that  $\gamma(G) \leq i(G) \leq \alpha(G)$ .

Another related notion is the (vertex) *coloring* of  $G$ , in which two adjacent vertices must be colored differently. Note that any subset of vertices colored the same in a coloring of  $G$  is necessarily an independent set. The *chromatic number*  $\chi(G)$  of  $G$  is the minimum number of colors in a coloring of  $G$ . Clearly,  $\alpha(G) \cdot \chi(G) \geq |V|$ .

---

\* Correspondence author.

The independence number  $\alpha(G)$  and the domination number  $\gamma(G)$  (and the chromatic number  $\chi(G)$ ) have received numerous studies due to their central roles in graph theory and theoretical computer science. Their exact values are NP-hard to compute [4], and hard to approximate. Raz and Safra showed that the domination number cannot be approximated within  $(1 - \epsilon) \log |V|$  for any fixed  $\epsilon > 0$ , unless  $\text{NP} \subset \text{DTIME}(|V|^{\log \log |V|})$  [9,3]; Zuckerman showed that neither the independence number nor the chromatic number can be approximated within  $|V|^{1-\epsilon}$  for any fixed  $\epsilon > 0$ , unless  $\text{P} = \text{NP}$  [14]; for  $i(G)$ , Halldórsson proved that it is also hard to approximate within  $|V|^{1-\epsilon}$  for any fixed  $\epsilon > 0$ , unless  $\text{NP} \subset \text{DTIME}(2^{o(|V|)})$  [5].

The above inapproximability results are for the worst case. For analyzing the average case performance of approximation algorithms, a probability distribution of the input graphs must be assumed and the most widely used distribution of graphs on  $n$  vertices is the random graph  $G(n, p)$ , which is a graph on  $n$  labeled vertices  $1, 2, \dots, n$ , and each edge is chosen to be an edge of  $G$  independently and with a probability  $p$ , where  $0 \leq p = p(n) \leq 1$ . A graph property holds *asymptotically almost surely* (a.a.s.) in  $G(n, p)$  if the probability that a graph drawn according to the distribution  $G(n, p)$  has the property tends to 1 as  $n$  tends to infinity [1].

Let  $\mathbb{L}n = \log_{1/(1-p)} n$ . Bollobás [2] and Łuczak [7] showed that a.a.s.  $\chi(G(n, p)) = (1 + o(1))n/\mathbb{L}n$  for a constant  $p$  and  $\chi(G(n, p)) = (1 + o(1))np/(2 \ln(np))$  for  $c/n \leq p(n) \leq o(1)$  where  $c$  is a constant. It follows from these results that a.a.s.  $\alpha(G(n, p)) = (1 - o(1))\mathbb{L}n$  for a constant  $p$  and  $\alpha(G(n, p)) = (1 - o(1))2 \ln(np)/p$  for  $C/n \leq p \leq o(1)$ . The greedy algorithm, which colors vertices of  $G(n, p)$  one by one and picks each time the first available color for a current vertex, is known to produce a.a.s. in  $G(n, p)$  with  $p \geq n^{\epsilon-1}$  a coloring whose number of colors is larger than the  $\chi(G(n, p))$  by only a constant factor (see Ch. 11 of the monograph of Bollobás [1]). Hence the largest color class produced by the greedy algorithm is a.a.s. smaller than  $\alpha(G(n, p))$  only by a constant factor.

For the domination number  $\gamma(G(n, p))$ , Wieland and Godbole showed that a.a.s. it is equal to either  $\lfloor \mathbb{L}n - \mathbb{L}(\lfloor \mathbb{L}n \rfloor (\ln n)) \rfloor + 1$  or  $\lfloor \mathbb{L}n - \mathbb{L}(\lfloor \mathbb{L}n \rfloor (\ln n)) \rfloor + 2$ , for a constant  $p$  or a suitable function  $p = p(n)$  [13]. It follows that a.a.s.  $i(G(n, p)) \geq \lfloor \mathbb{L}n - \mathbb{L}(\lfloor \mathbb{L}n \rfloor (\ln n)) \rfloor + 1$ . Recently, Wang proved for  $i(G(n, p))$  an a.a.s. upper bound of  $\lfloor \mathbb{L}n - \mathbb{L}(\lfloor \mathbb{L}n \rfloor (\ln n)) \rfloor + k + 1$ , where  $k = k(p) \geq 1$  is the smallest integer satisfying  $(1 - p)^k < \frac{1}{2}$  [12].

Average case performance analysis of an approximation algorithm over random instances could be inconclusive, because the random instances usually have very special properties that distinguish them from real-world instances. For instance, for a constant  $p$ , the random graph  $G(n, p)$  is expected to be dense. On the other hand, an approximation algorithm performs very well on most random instances can fail miserably on some “hard” instances. For instance, it has been shown by Kučera [6] that for any fixed  $\epsilon > 0$  there exists a graph  $G$  on  $n$  vertices for which, even after a random permutation of vertices, the greedy algorithm produces a.a.s. a coloring using at least  $n / \log_2 n$  colors, while  $\chi(G) \leq n^\epsilon$ . To

overcome this, Spielman and Teng [10] introduced the smoothed analysis. This new analysis is a hybrid of the worst-case and the average-case analyses, and it inherits the advantages of both, by measuring the expected performance of the algorithm under slight random perturbations of the worst-case inputs. If the smoothed complexity of an algorithm is low, then it is unlikely that the algorithm will take long time to solve practical instances whose data are subject to slight noises and imprecision. Though the smoothed analysis concept was introduced for the complexity of algorithms, we extend its idea to depict the essential properties of problems.

In this paper, we study the approximability of the minimum independent dominating set (MIDS) problem under the smoothed analysis, and we present a simple deterministic greedy algorithm beating the strong inapproximability bound of  $n^{1-\epsilon}$ , with polynomial expected running time. Our probabilistic model is the smoothed extension of random graph  $G(n, p)$  (also called semi-random graphs in [8]), proposed by Spielman and Teng [11]: given a graph  $G = (V, E)$ , we define its perturbed graph  $\mathbf{g}(G, p)$  by negating the existence of edges independently with a probability of  $p > 0$ . That is,  $\mathbf{g}(G, p)$  has the same vertex set  $V$  as  $G$  but it contains edge  $e$  with probability  $p_e$ , where  $p_e = 1 - p$  if  $e \in E$  or otherwise  $p_e = p$ . For sufficiently large  $p$ , Manthey and Plociennik presented an algorithm approximating the independence number  $\alpha(\mathbf{g}(G, p))$  with a worst-case performance ratio  $O(\sqrt{np})$  and with polynomial expected running time [8].

Re-define  $\mathbb{L}n = \log_{1/p} n$ . We first prove on  $\gamma(\mathbf{g}(G, p))$ , and thus on  $i(\mathbf{g}(G, p))$  as well, an a.a.s. lower bound of  $\mathbb{L}n - \mathbb{L}((\mathbb{L}n)(\ln n))$  if  $p > \frac{1}{n}$ . We then prove on  $\alpha(\mathbf{g}(G, p))$ , and thus on  $i(\mathbf{g}(G, p))$  as well, an a.a.s. upper bound of  $2 \ln n / p$  if  $p < \frac{1}{2}$  or  $2 \ln n / (1 - p)$  otherwise. Given that the a.a.s. values of  $\alpha(G(n, p))$  and  $i(G(n, p))$  in random graph  $G(n, p)$ , our upper bound comes with no big surprise; nevertheless, our upper bound is derived by a direct counting process which might be interesting by itself. Furthermore, we extend our counting techniques to prove on  $i(\mathbf{g}(G, p))$  a tail bound that, when  $4 \ln^2 n / n < p \leq \frac{1}{2}$ ,  $\Pr[i(\mathbf{g}(G, p)) \geq \sqrt{4n/p}] \leq 2^{-n}$ . We then present a simple greedy algorithm to approximate  $i(\mathbf{g}(G, p))$ , and prove that its worst case performance ratio is  $\sqrt{4n/p}$  and its expected running time is polynomial.

## 2 A.a.s. Bounds on the Independent Domination Number

We need the following several facts.

**Fact 1.**  $e^{\frac{x}{1+x}} \leq 1 + x \leq e^x$  holds for all  $x \in [-1, 1]$ .

**Fact 2.**  $\left(\frac{n}{r}\right)^r \leq \binom{n}{r} \leq \left(\frac{ne}{r}\right)^r$  holds for all  $r = 0, 1, 2, \dots, n$ .

**Fact 3.** (Jensen's Inequality) For a real convex function  $f(x)$ , numbers  $x_1, x_2, \dots, x_n$  in its domain, and positive weights  $a_i$ ,  $f\left(\frac{\sum a_i x_i}{\sum a_i}\right) \leq \frac{\sum a_i f(x_i)}{\sum a_i}$ ; the inequality is reversed if  $f(x)$  is concave.

Given any graph  $G = (V, E)$ , let  $\mathbf{g}(G, p)$  denote its perturbed graph, which has the same vertex set  $V$  as  $G$  and contains edge  $e$  with a probability of

$$p_e = \begin{cases} 1 - p, & \text{if } e \in E, \\ p, & \text{otherwise.} \end{cases}$$

## 2.1 An a.a.s. Lower Bound

Recall that  $\gamma(\mathbf{g}(G, p))$  and  $i(\mathbf{g}(G, p))$  are the domination number and the independent domination number of  $\mathbf{g}(G, p)$ , respectively. Also,  $\mathbb{L}n = \log_{1/p} n$ .

**Theorem 1.** *For any graph  $G = (V, E)$  and  $\frac{1}{n} < p \leq 1$ , a.a.s.*

$$\gamma(\mathbf{g}(G, p)) \geq \mathbb{L}n - \mathbb{L}((\mathbb{L}n)(\ln n)).$$

*Proof.* Let  $\mathcal{S}_r$  be the collection of all  $r$ -subsets of vertices in  $\mathbf{g}(G, p)$ , and these  $\binom{n}{r}$  sets of  $\mathcal{S}_r$  are ordered in some way. Define  $I_j^r$  as a boolean variable to indicate whether or not the  $j$ -th  $r$ -subset of  $\mathcal{S}_r$ ,  $V_j$ , is a dominating set; set  $X_r = \sum_j I_j^r$ .

Clearly,  $\gamma(\mathbf{g}(G, p)) < r$  implies that there are size- $r$  dominating sets. Therefore,

$$\Pr[\gamma(\mathbf{g}(G, p)) < r] \leq \Pr[X_r \geq 1] \leq E(X_r),$$

where  $E(X_r)$  is the expected value of  $X_r$ . (We abuse the notation  $E$  a little, but its meaning should be clear at every occurrence.)

For the  $j$ -th  $r$ -subset  $V_j$ , let  $E_j$  be the subset of induced edges on  $V_j$  from the original graph  $G = (V, E)$ ; let  $V_j^c = V - V_j$ , the complement subset of vertices. Also, for each vertex  $u \in V_j^c$ , define  $E(u, V_j) = \{(u, v) \in E \mid v \in V_j\}$ , and its size  $n_{uj} = |E(u, V_j)|$ . Using Fact 1, we can estimate  $E(X_r)$  as follows:

$$\begin{aligned} E(X_r) &= \sum_{j=1}^{\binom{n}{r}} E(I_j^r) = \sum_{j=1}^{\binom{n}{r}} \prod_{u \in V_j^c} \left( 1 - \prod_{v \in V_j} (1 - p_{(u,v)}) \right) \\ &\leq \sum_{j=1}^{\binom{n}{r}} \prod_{u \in V_j^c} \exp \left( - \prod_{v \in V_j} (1 - p_{(u,v)}) \right) \\ &= \sum_{j=1}^{\binom{n}{r}} \exp \left( - \sum_{u \in V_j^c} \prod_{v \in V_j} (1 - p_{(u,v)}) \right) \\ &= \sum_{j=1}^{\binom{n}{r}} \exp \left( - \sum_{u \in V_j^c} p^{n_{uj}} (1-p)^{r-n_{uj}} \right) \\ &= \sum_{j=1}^{\binom{n}{r}} \exp \left( - \sum_{u \in V_j^c} \left( \frac{p}{1-p} \right)^{n_{uj}} (1-p)^r \right). \end{aligned}$$

Since function  $f(x) = (\frac{p}{1-p})^x$  is convex in the domain  $[0, n]$ , by Jensen's Inequality, the above becomes

$$E(X_r) \leq \sum_{j=1}^{\binom{n}{r}} \exp \left( - \left( \frac{p}{1-p} \right)^{\frac{1}{n-r}} \sum_{u \in V_j^c} n_{uj} (n-r)(1-p)^r \right).$$

Since function  $g(x) = e^{-ax^b}$  with  $a = (\frac{p}{1-p})^{\frac{1}{n-r}}$  and  $b = (n-r)(1-p)^r$  is concave in the domain  $[0, n^2]$ , again by Jensen's Inequality, we further have

$$E(X_r) \leq \binom{n}{r} \exp \left( - \left( \frac{p}{1-p} \right)^{\frac{1}{(n-r)\binom{n}{r}}} \sum_{j=1}^{\binom{n}{r}} \sum_{u \in V_j^c} n_{uj} (n-r)(1-p)^r \right). \quad (1)$$

Recall that  $n_{uj}$  is number of edges in the original graph  $G = (V, E)$  between  $u$  and vertices of  $V_j$ . Each edge  $e \in E$  is thus counted towards the quantity  $\left( \sum_{j=1}^{\binom{n}{r}} \sum_{u \in V_j^c} n_{uj} \right)$  exactly  $2\binom{n-2}{r-1}$  times. That is,

$$\sum_{j=1}^{\binom{n}{r}} \sum_{u \in V_j^c} n_{uj} = 2\binom{n-2}{r-1} |E| = \frac{\binom{n}{r} r(n-r)|E|}{\binom{n}{2}}. \quad (2)$$

Using Eq. (2), Fact 2 and  $r = \mathbb{L}n - \mathbb{L}((\mathbb{L}n)(\ln n))$ , Eq. (1) becomes

$$\begin{aligned} E(X_r) &\leq \binom{n}{r} \exp \left( - \left( \frac{p}{1-p} \right)^{\frac{r|E|}{\binom{n}{2}}} (n-r)(1-p)^r \right) \\ &\leq \binom{n}{r} \exp \left( - \left( \frac{p}{1-p} \right)^r (n-r)(1-p)^r \right) \\ &\leq \binom{ne}{r}^r \exp(-p^r(n-r)) \\ &\leq \exp \left( r \ln n + r - r \ln r - \frac{(\mathbb{L}n)(\ln n)}{n} (n-r) \right) \\ &= \exp((\mathbb{L}n)(\ln n) - \mathbb{L}((\mathbb{L}n)(\ln n)) \ln n + r - r \ln r \\ &\quad - (\mathbb{L}n)(\ln n) + r(\mathbb{L}n)(\ln n)/n) \\ &= \exp(-\mathbb{L}((\mathbb{L}n)(\ln n)) \ln n - r(\ln r - (\mathbb{L}n)(\ln n)/n - 1)) \\ &\leq \exp(-\mathbb{L}((\mathbb{L}n)(\ln n)) \ln n - r(\ln r - 2)). \end{aligned} \quad (3)$$

The right hand side in Eq. (3) approaches 0 when  $n \rightarrow +\infty$ . Since  $p > \frac{1}{n}$  guarantees  $r \geq 1$ ,  $\mathbb{L}n - \mathbb{L}((\mathbb{L}n)(\ln n))$  is an a.a.s. lower bound on  $\gamma(\mathbf{g}(G, p))$ . This proves the theorem.  $\square$

Since  $\Pr[i(\mathbf{g}(G, p)) < r] \leq \Pr[\gamma(\mathbf{g}(G, p)) < r]$ , we have the following corollary:

**Corollary 1.** *For any graph  $G = (V, E)$  and  $\frac{1}{n} < p \leq 1$ , a.a.s.*

$$i(\mathbf{g}(G, p)) \geq \mathbb{L}n - \mathbb{L}((\mathbb{L}n)(\ln n)).$$

## 2.2 An a.a.s. Upper Bound

Recall that  $\alpha(\mathbf{g}(G, p))$  is the independence number of  $\mathbf{g}(G, p)$ .

**Theorem 2.** *For any graph  $G = (V, E)$ , a.a.s.*

$$\alpha(\mathbf{g}(G, p)) \leq \begin{cases} \frac{2 \ln n}{p}, & \text{if } p \in (\frac{2 \ln n}{n}, \frac{1}{2}], \\ \frac{2 \ln n}{1-p}, & \text{if } p \in [\frac{1}{2}, 1 - \frac{2 \ln n}{n}). \end{cases}$$

*Proof.* Let  $\mathcal{S}_r$  be the collection of all  $r$ -subsets of vertices in  $\mathbf{g}(G, p)$ , and these  $\binom{n}{r}$  sets of  $\mathcal{S}_r$  are ordered in some way. Define  $I_j^r$  as a boolean variable to indicate whether or not the  $j$ -th  $r$ -subset of  $\mathcal{S}_r$  is an independent set; set  $X_r = \sum_j I_j^r$ . Since  $\alpha(\mathbf{g}(G, p)) > r$  implies that there is at least one independent  $r$ -subset, i.e.  $X_r > 0$ , the probability of the event  $\alpha(\mathbf{g}(G, p)) > r$  is less than or equal to the probability of the event  $X_r > 0$ , i.e.

$$\Pr[\alpha(\mathbf{g}(G, p)) > r] \leq \Pr[X_r > 0].$$

On the other hand, let  $A_j^r$  denote the event  $I_j^r = 0$ , i.e. the  $j$ -th  $r$ -subset is not independent. It follows that  $X_r = 0$  is equivalent to the joint event  $\cap_j A_j^r$ , i.e.

$$\Pr[X_r = 0] = \Pr[\cap_j A_j^r] \geq \prod_j \Pr[A_j^r] = \prod_j (1 - \Pr[I_j^r = 1]).$$

Therefore, we have

$$\Pr[\alpha(\mathbf{g}(G, p)) > r] \leq 1 - \prod_j (1 - \Pr[I_j^r = 1]). \quad (4)$$

Let  $E_j^r$  denote the subset of edges of  $\mathbf{g}(G, p)$ , each of which connects two vertices in the  $j$ -th  $r$ -subset of  $\mathcal{S}_r$ . Note that  $|E_j^r| \in [0, \binom{r}{2}]$ . Among all the edges of  $E_j^r$ , assume there are  $n_j^r$  of them coming from the original edge set  $E$  of  $G$ . It follows that

$$\Pr[I_j^r = 1] = \prod_{e \in E_j^r} (1 - p_e) = \left( \frac{p}{1-p} \right)^{n_j^r} (1-p)^{\binom{r}{2}}.$$

Using this and Fact 1 in Eq. (4) gives us

$$\Pr[\alpha(\mathbf{g}(G, p)) > r] \leq 1 - \prod_j (1 - \Pr[I_j^r = 1])$$

$$\begin{aligned}
&\leq 1 - \prod_{j=1}^{\binom{n}{r}} \exp \left( -\frac{\Pr[I_j^r = 1]}{1 - \Pr[I_j^r = 1]} \right) \\
&= 1 - \exp \left( -\sum_{j=1}^{\binom{n}{r}} \frac{\Pr[I_j^r = 1]}{1 - \Pr[I_j^r = 1]} \right) \\
&= 1 - \exp \left( -\sum_{j=1}^{\binom{n}{r}} \frac{\left(\frac{p}{1-p}\right)^{n_j^r} (1-p)^{\binom{r}{2}}}{1 - \left(\frac{p}{1-p}\right)^{n_j^r} (1-p)^{\binom{r}{2}}} \right). \quad (5)
\end{aligned}$$

Consider the function  $f(x) = \frac{a^x b}{1-a^x b}$  in Eq. (5), where  $a = \frac{p}{1-p} > 0$ ,  $b = (1-p)^{\binom{r}{2}} \in (0, 1)$ , and  $0 \leq x \leq \binom{r}{2}$ . Since its derivative

$$f'(x) = \frac{a^x b \ln a}{(1-a^x b)^2} \begin{cases} < 0, & \text{if } a < 1, \\ = 0, & \text{if } a = 1, \\ > 0, & \text{if } a > 1, \end{cases}$$

$f(x)$  is strictly decreasing if  $a < 1$ , or strictly increasing if  $a > 1$ . Therefore, the maximum value of function  $f(x)$  is achieved at  $x = 0$  if  $a \leq 1$ , or at  $x = \binom{r}{2}$  if  $a \geq 1$ .

When  $p \leq \frac{1}{2}$ , that is  $a = \frac{p}{1-p} \leq 1$ , Eq. (5) becomes

$$\begin{aligned}
\Pr[\alpha(\mathbf{g}(G, p)) > r] &\leq 1 - \exp \left( -\sum_{j=1}^{\binom{n}{r}} \frac{(1-p)^{\binom{r}{2}}}{1 - (1-p)^{\binom{r}{2}}} \right) \\
&= 1 - \exp \left( -\binom{n}{r} \frac{(1-p)^{\binom{r}{2}}}{1 - (1-p)^{\binom{r}{2}}} \right). \quad (6)
\end{aligned}$$

To prove  $\Pr[\alpha(\mathbf{g}(G, p)) > r] \rightarrow 0$  as  $n \rightarrow +\infty$ , we only need to prove that  $\binom{n}{r} \frac{(1-p)^{\binom{r}{2}}}{1 - (1-p)^{\binom{r}{2}}} \rightarrow 0$  as  $n \rightarrow +\infty$ . Using Fact 2, we have

$$\binom{n}{r} \frac{(1-p)^{\binom{r}{2}}}{1 - (1-p)^{\binom{r}{2}}} = \frac{\binom{n}{r}}{\left(\frac{1}{1-p}\right)^{\binom{r}{2}} - 1} \leq \frac{\left(\frac{ne}{r}\right)^r}{\left(\frac{1}{1-p}\right)^{\binom{r}{2}} - 1}. \quad (7)$$

Setting  $r = 2 \ln n/p$ . We see that  $r \rightarrow +\infty$  as  $n \rightarrow +\infty$ . On the other hand, when  $r$  is large enough, we have

$$\left(\frac{1}{1-p}\right)^{\binom{r}{2}} - 1 = \left(\frac{1}{1-p}\right)^{\binom{r}{2}} (1 - o(1)). \quad (8)$$

Using Eq. (8) and Fact 1, when  $n$  is sufficiently large, Eq. (7) becomes

$$\begin{aligned}
\binom{n}{r} \frac{(1-p)^{\binom{r}{2}}}{1 - (1-p)^{\binom{r}{2}}} &\leq \frac{\left(\frac{ne}{r}\right)^r}{\left(\frac{1}{1-p}\right)^{\binom{r}{2}}} (1 + o(1)) = \left(\frac{ne}{r \left(\frac{1}{1-p}\right)^{\frac{r-1}{2}}}\right)^r (1 + o(1)) \\
&= \left(\frac{ne}{r \left(1 + \frac{p}{1-p}\right)^{\frac{r-1}{2}}}\right)^r (1 + o(1)) \\
&\leq \left(\frac{ne}{r \exp\left(\frac{p}{1+\frac{p}{1-p}} \cdot \frac{r-1}{2}\right)}\right)^r (1 + o(1)) \\
&= \left(\frac{ne}{r \exp(p \cdot \frac{r-1}{2})}\right)^r (1 + o(1)) \\
&= \left(\frac{ne^{1+\frac{p}{2}}}{re^{\frac{rp}{2}}}\right)^r (1 + o(1)) \\
&= \left(\frac{e^{1+\frac{p}{2}}}{r}\right)^r (1 + o(1)) \\
&\leq \left(\frac{e^{\frac{5}{4}}}{r}\right)^r (1 + o(1)). \tag{9}
\end{aligned}$$

The quantity  $\left(\frac{e^{\frac{5}{4}}}{r}\right)^r$  in Eq. (10) is less than  $0.5^r$  when  $n$  is sufficiently large, the latter approaches 0 when  $n \rightarrow +\infty$ . This proves that when  $p \leq \frac{1}{2}$ ,  $\Pr[\alpha(\mathbf{g}(G, p)) > r] \rightarrow 0$  as  $n \rightarrow +\infty$ . That is, when  $p \leq \frac{1}{2}$ , a.a.s.  $\alpha(\mathbf{g}(G, p)) \leq 2 \ln n / p$ .

When  $p \geq \frac{1}{2}$ , that is  $a = \frac{p}{1-p} \geq 1$ ,  $q = 1 - p \leq \frac{1}{2}$  and exactly the same argument as when  $p \leq \frac{1}{2}$  applies by replacing  $p$  with  $1 - q$ , which shows that a.a.s.  $\alpha(\mathbf{g}(G, p)) \leq 2 \ln n / (1 - p)$ . This proves the theorem.  $\square$

Since  $\alpha(\mathbf{g}(G, p)) \geq i(\mathbf{g}(G, p))$ ,  $\Pr[i(\mathbf{g}(G, p)) > r] \leq \Pr[\alpha(\mathbf{g}(G, p)) > r]$  and thus we have the following corollary:

**Corollary 2.** *For any graph  $G = (V, E)$ , a.a.s.*

$$i(\mathbf{g}(G, p)) \leq \begin{cases} \frac{2 \ln n}{p}, & \text{if } p \in (\frac{2 \ln n}{n}, \frac{1}{2}], \\ \frac{2 \ln n}{1-p}, & \text{if } p \in [\frac{1}{2}, 1 - \frac{2 \ln n}{n}). \end{cases}$$

### 3 A Tail Bound on the Independent Domination Number

**Theorem 3.** *For any graph  $G = (V, E)$  and  $p \in (\frac{4 \ln^2 n}{n}, \frac{1}{2}]$ ,*

$$\Pr[i(\mathbf{g}(G, p)) \geq \sqrt{\frac{4n}{p}}] \leq \Pr[\alpha(\mathbf{g}(G, p)) \geq \sqrt{\frac{4n}{p}}] \leq 2^{-n}.$$

*Proof.* The proof of this theorem flows exactly the same of the proof of Theorem 2. In fact, with  $p \leq \frac{1}{2}$ , we have both Eq. (6) and Eq. (7) hold. Different from the proof of Theorem 2 where  $r = 2 \ln n/p$ , we have now  $r = \sqrt{\frac{4n}{p}} \geq 2 \ln n/p$  and therefore Eq. (8) holds as well. Again, using Eq. (8) and Fact 1, when  $n$  is sufficiently large, Eq. (9) still holds. It then follows from Fact 1 that Eq. (6) becomes

$$\begin{aligned} \Pr[i(\mathbf{g}(G, p)) \geq r] &\leq \Pr[\alpha(\mathbf{g}(G, p)) \geq r] \\ &\leq 1 - \exp\left(-\left(\frac{ne^{1+\frac{p}{2}}}{re^{\frac{rp}{2}}}\right)^r (1 + o(1))\right). \end{aligned} \quad (11)$$

Using  $r = \sqrt{\frac{4n}{p}}$ , we prove in the following that  $\left(\frac{ne^{1+\frac{p}{2}}}{re^{\frac{rp}{2}}}\right)^r (1 + o(1)) = o(1)$ . And consequently by Fact 1 again and  $r = \sqrt{\frac{4n}{p}} \geq \sqrt{8n}$ , Eq. (11) becomes

$$\begin{aligned} \Pr[i(\mathbf{g}(G, p)) \geq r] &\leq \left(\frac{ne^{1+\frac{p}{2}}}{re^{\frac{rp}{2}}}\right)^r (1 + o(1)) \leq \frac{e}{2} \left(\frac{ne^{1+\frac{p}{2}}}{re^{\frac{rp}{2}}}\right)^r \\ &= \frac{e}{2} \exp\left(-r\left(\ln r + \frac{1}{2}rp - \ln n - 1 - \frac{p}{2}\right)\right) \\ &= \frac{e}{2} \exp\left(-r\left(\ln r + \frac{1}{4}rp - \ln n - 1 - \frac{p}{2}\right) - \frac{1}{4}r^2p\right) \\ &= \frac{e}{2} \exp\left(-r\left(\ln r + \frac{1}{4}rp - \ln n - 1 - \frac{p}{2}\right) - n\right). \end{aligned} \quad (12)$$

The quantity  $(\ln r + \frac{1}{4}rp - \ln n - 1 - \frac{p}{2})$  in Eq. (12) is non-negative when  $n \geq 2$ , since

$$\begin{aligned} \ln r + \frac{1}{4}rp - \ln n - 1 - \frac{p}{2} &\geq \frac{1}{2} \ln(8n) + \frac{1}{4} \sqrt{4np} - \ln n - 1 - \frac{1}{4} \\ &\geq \frac{1}{2} \ln(8n) + \frac{1}{4} \sqrt{4n \cdot \frac{4 \ln^2 n}{n}} - \ln n - 1 - \frac{1}{4} \\ &= \frac{1}{2} \left(\ln(8n) - \frac{5}{2}\right) \geq 0. \end{aligned}$$

It follows that Eq. (12) becomes

$$\begin{aligned} \Pr[i(\mathbf{g}(G, p)) \geq r] &\leq \frac{e}{2} \exp\left(-r\left(\ln r + \frac{1}{4}rp - \ln n - 1 - \frac{p}{2}\right) - n\right) \\ &\leq \frac{e}{2} e^{-n} < 2^{-n}. \end{aligned}$$

This proves the theorem.  $\square$

## 4 Approximating the Independent Domination Number

We present next a simple algorithm, denoted as *Approx-IDS*, for computing an independent dominating set in  $\mathfrak{g}(G, p)$ . In the first phase, algorithm Approx-IDS repeatedly picks a maximum degree vertex and updates the graph by deleting the picked vertex and all its neighbors; it terminates when there is no more vertex and returns a subset  $I$  of  $V$ . If  $|I| \leq \sqrt{\frac{4n}{p}}$ , algorithm Approx-IDS terminates and outputs  $I$ ; otherwise it moves into the second phase. In the second phase, algorithm Approx-IDS performs an exhaustive search over all subsets of  $V$ , and returns the minimum independent dominating set  $I^*$ .

**Theorem 4.** *For any graph  $G = (V, E)$  and  $p \in (\frac{4\ln^2 n}{n}, \frac{1}{2}]$ , algorithm Approx-IDS is a  $\sqrt{\frac{4n}{p}}$ -approximation to the MIDS problem on the perturbed graph  $\mathfrak{g}(G, p)$ , and it has polynomial expected running time.*

*Proof.* Note that  $i(\mathfrak{g}(G, p)) \geq 1$ . The subset  $I$  of  $V$  computed by algorithm Approx-IDS is a dominating set, since every vertex of  $V$  is either in  $I$ , or is a neighbor of some vertex in  $I$ . Also, no two vertices of  $I$  can be adjacent, since otherwise one would be removed in the iteration its neighbor was picked by the algorithm. Therefore,  $I$  is an independent dominating set of  $\mathfrak{g}(G, p)$ . It follows that if algorithm Approx-IDS terminates after the first phase,  $|I| \leq \sqrt{\frac{4n}{p}} \cdot i(\mathfrak{g}(G, p))$ . Also clearly the first phase takes  $O(n^3)$  time.

In the second phase, a maximum of  $2^n$  subsets of  $V$  are examined by the algorithm. Since checking each of them to be an independent dominating set or not takes no more than  $O(n^2)$  time, the overall running time is  $O(2^n n^2)$ . Note that this phase returns  $I^*$  with  $|I^*| = i(\mathfrak{g}(G, p))$ . As  $\alpha(\mathfrak{g}(G, p)) \geq |I| > \sqrt{\frac{4n}{p}}$ , Theorem 3 tells that the probability of executing this second phase is no more than  $2^{-n}$ . Therefore, the expected running time of the second phase is  $O(n^2)$ . This proves the theorem.  $\square$

## 5 Conclusions

We have performed a smooth analysis for approximating the minimum independent dominating set problem. The probabilistic model we used is the perturbed graph  $\mathfrak{g}(G, p)$  of the input graph  $G = (V, E)$  [11]. We have proved a.a.s. bounds and a tail bound on the independent domination number of  $\mathfrak{g}(G, p)$ , and presented an algorithm with the worst-case performance ratio of  $\sqrt{\frac{4|V|}{p}}$  and with polynomial expected running time.

## References

1. Bollobás, B.: Random Graphs. Academic Press, New York (1985)

2. Bollobás, B.: The chromatic number of random graphs. *Combinatorica* 8, 49–55 (1988)
3. Feige, U.: A threshold of for approximating set cover. *Journal of the ACM* 45, 634–652 (1998)
4. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. W. H. Freeman and Company, San Francisco (1979)
5. Halldórsson, M.M.: Approximating the minimum maximal independence number. *Information Processing Letters* 46, 169–172 (1993)
6. Kučera, L.: The greedy coloring is a bad probabilistic algorithm. *Journal of Algorithms* 12, 674–684 (1991)
7. Łuczak, T.: The chromatic number of random graphs. *Combinatorica* 11, 45–54 (1991)
8. Manthey, B., Plociennik, K.: Approximating independent set in perturbed graphs. *Discrete Applied Mathematics* (2012) (in press)
9. Raz, R., Safra, S.: A sub-constant error-probability low-degree test, and sub-constant error-probability PCP characterization of NP. In: Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC), pp. 475–484 (1997)
10. Spielman, D.A., Teng, S.-H.: Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM* 51, 385–463 (2004)
11. Spielman, D.A., Teng, S.-H.: Smoothed analysis: an attempt to explain the behavior of algorithms in practice. *Communications of the ACM* 52, 76–84 (2009)
12. Wang, C.: The independent domination number of random graph. *Utilitas Mathematica* 82, 161–166 (2010)
13. Wieland, B., Godbole, A.P.: On the domination number of a random graph. *The Electronic Journal of Combinatorics* 8, #R37 (2001)
14. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing* 3, 103–128 (2007)

# A Linear-Time Algorithm for the Minimum Degree Hypergraph Problem with the Consecutive Ones Property

Chih-Hsuan Li, Jhih-Hong Ye, and Biing-Feng Wang

Department of Computer Science, National Tsing Hua University  
Hsinchu, Taiwan 30013, Republic of China  
`{chhsuan, jhong, jhong}@cs.nthu.edu.tw`

**Abstract.** Given a set  $S$ , two collections  $C_r$  and  $C_b$  of non-empty subsets of  $S$  and a positive integer  $k < |S|$ , the minimum degree hypergraph (MDH) problem is to find a subset  $S'$  of  $S$  such that  $S' \cap B \neq \emptyset$  for all  $B \in C_b$  and  $|S' \cap R| \leq k$  for all  $R \in C_r$ . This paper presents a linear-time algorithm for the MDH problem with  $C_r \cup C_b$  having the consecutive ones property. The presented algorithm improves the previous upper bound from  $O(|S|^2)$ .

## 1 Introduction

Due to practical considerations, numerous generalizations of the set cover problem have been defined and studied [4, 5, 7, 8, 11, 12]. The minimum degree hypergraph (MDH) problem and the red-blue set cover (RBSC) problem are two examples. Let  $S$  be a set and let  $C_b$  (blue collection) and  $C_r$  (red collection) be two collections of subsets of  $S$ . The MDH problem is to find a subset  $S' \subseteq S$  such that  $S' \cap B \neq \emptyset$  for all  $B \in C_b$  and  $|S' \cap R| \leq k$  for all  $R \in C_r$ , where  $k < |S|$  is a given positive integer. The RBSC problem is to find a subset  $S' \subseteq S$  with  $S' \cap B \neq \emptyset$  for all  $B \in C_b$  which minimizes  $|\{R | R \in C_r, S' \cap R \neq \emptyset\}|$ . Feder *et al.* [8] introduced the MDH problem and gave a polynomial-time approximation algorithm that has an approximation ratio of  $O(\lg |S|)$ . Kuhn *et al.* [9] studied a special case of the MDH problem, in which  $C_r = C_b$ , and showed that it has similar approximation properties as the classical set cover problem. The RBSC problem was introduced by Carr *et al.* [5]. They provided several positive and negative results concerning the polynomial-time approximability of the RBSC problem.

Since the set cover problem is NP-complete, one may only hope to find efficient algorithms for special cases of practical interest. A famous case is the set cover problem with the *consecutive ones property* (C1P), which means that the elements of  $S$  can be arranged in a linear order such that each set in  $C$  contains consecutive elements of  $S$ . This special case admits a polynomial-time solution, a fact which is utilized in many practical applications [10–13]. Recently, Dom *et al.* [7] studied the MDH and the RBSC problems with the C1P. Their study is motivated by applications in geographic background which may have the C1P

or be “close” to the C1P [9–11]. They proved several NP-completeness results in case that at most one of  $C_b$  and  $C_r$  has the C1P. And, they gave efficient algorithms for the MDH and the RBSC problems with  $C_r \cup C_b$  having the C1P. In addition, they gave an efficient algorithm for the MDH problem with  $k = 1$  and all sets in  $C_b$  having size two. For the RBSC problem with the C1P, Dom *et al.*’s algorithm requires  $O(|C_b||C_r||S|^2)$  time. Chang *et al.* [6] improved this result to  $O(|C_b||S| + |C_r||S| + |S|^2)$ . Later, Wang and Li [14] further reduced this upper bound to  $O(|C_b| + |C_r|\lg|S| + |S|\lg|S|)$ . For the MDH problem with the C1P, Dom *et al.*’s algorithm requires  $O(|C_b||S| + |C_r||S| + |S|^2)$  time. For the MDH problem with  $k = 1$  and subset size two, Dom *et al.*’s algorithm requires  $O(|S| + |C_b| + \sum_{R \in C_r} |R|^2)$  time. Wang and Li [14] gave an improved linear-time algorithm.

**Contribution:** The focus of this paper is the MDH problem with  $C_r \cup C_b$  having the C1P. For this problem, Dom *et al.*’s algorithm requires  $O(|S|^2)$  time. In this paper, we first characterize two special cases of this problem, where the first is obviously infeasible, and the second is feasible and admits a simple linear-time algorithm. Then, we show that any input instance can be reduced to either the first or the second special case. Finally, we give a sophisticated algorithm that performs the reduction in linear time. Consequently, we obtain an optimal algorithm for the MDH problem with  $C_r \cup C_b$  having the C1P.

**Table 1.** Results on the MDH problem with the C1P

Time	Remark
$O( S ^2 +  C_b  S  +  C_r  S )$	[7]
$O( S  +  C_b  +  C_r )$	this paper

Section 2 gives notation and definitions. Section 3 presents an  $O(|S|^2)$ -time algorithm. Section 4 reduces the time complexity of the algorithm in Section 3 to  $O(|S|\alpha(|S|))$ , where  $\alpha$  is the inverse Ackermann’s function. Section 5 further reduces the time complexity to  $O(|S| + |C_b| + |C_r|)$ .

## 2 Notation and Preliminaries

Let  $S$  be a set of  $n$  elements. Without loss of generality, assume that the elements of  $S$  are already sorted in a linear order such that each set in  $C_r \cup C_b$  contains consecutive elements of  $S$ . In case this is not true, such a linear order can be found in  $O(|S| + \sum_{A \in C_r \cup C_b} |A|)$  time [3]. For ease of presentation, the  $i$ th element in  $S$  is simply denoted by  $i$ ,  $1 \leq i \leq n$ . Assume that each set  $\{i, i+1, \dots, j\} \in C_r \cup C_b$  is given by an interval  $[i, j]$ . Each interval in  $C_b$  is called a *blue* interval and each interval in  $C_r$  is called a *red* interval. We use  $left(I)$  and  $right(I)$ , respectively, to denote the left and right endpoints of an interval  $I$ .

Let  $S'$  be a subset of  $S$ . It *covers* a blue interval  $B$  if  $S' \cap B \neq \emptyset$ , and it *hits* a red interval  $R$   $m$  times if  $|S' \cap R| = m$ . If there exists a subset  $S' \subseteq S$  that covers all intervals in  $C_b$  and hits each red interval in  $C_r$  at most  $k$  times, then  $(C_r, C_b)$  is *feasible*; otherwise,  $(C_r, C_b)$  is *infeasible*. If a blue interval  $B$  contains another blue interval  $B'$ , it can be removed from  $C_b$ , since any subset that covers  $B'$  also covers  $B$ . Similarly, if a red interval  $R$  is a subinterval of another red interval  $R'$ , it can be removed from  $C_r$ , since any subset that hits  $R'$  at most  $k$  times also hits  $R$  at most  $k$  times. Thus, we have the following.

**Lemma 1.** If a blue interval contains another blue interval, it can be removed from  $C_b$  without changing the feasibility of  $(C_r, C_b)$ . If a red interval is contained in another red interval, it can be removed from  $C_r$  without changing the feasibility of  $(C_r, C_b)$ .

A collection of intervals has the *non-nested property* if no interval is contained in another. Two instances  $(C_r, C_b)$  and  $(C'_r, C'_b)$  of the MDH problem are *equivalent* if they have the same feasible solutions. By Lemma 1, we can reduce a given  $(C_r, C_b)$  into an equivalent instance in which the blue and red collections both satisfy the non-nested property. It is easy to do the reduction in linear time. In the remainder of this paper, we assume that both  $C_b$  and  $C_r$  satisfy the non-nested property. Consequently, both  $|C_b|$  and  $|C_r|$  are not larger than  $n$ . In addition, we assume that the intervals in  $C_b$  and  $C_r$  are, respectively, ordered by increasing left endpoints. We use  $B_i$  to denote the  $i$ th interval in  $C_b$  and use  $R_j$  to denote the  $j$ th interval in  $C_r$ .

### 3 A Two-Phase Algorithm

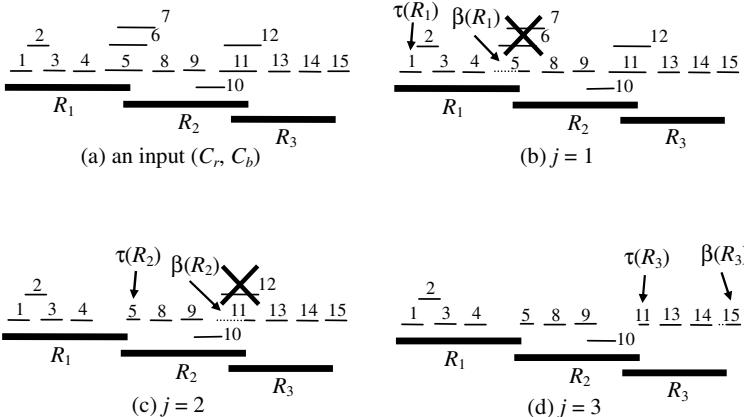
#### 3.1 The Framework

We start with the following simple observation.

**Lemma 2.** If there is an interval  $R$  in  $C_r$  that contains  $k + 1$  pairwise disjoint intervals in  $C_b$ , then  $(C_r, C_b)$  is infeasible.

The intervals in  $C_r$  are classified into three types: witnesses of infeasibility, critical intervals, and non-critical intervals. Consider an interval  $R \in C_r$ . If  $R$  contains a set of  $k + 1$  pairwise disjoint intervals in  $C_b$ , then  $R$  is a *witness of infeasibility* (or, simply a *witness*). If  $R$  is not a witness but there exist  $k + 1$  pairwise disjoint intervals in  $C_b$  whose left endpoints are contained in  $R$ , then  $R$  is a *critical interval*. Note that if  $R$  contains the left endpoints of  $k + 1$  pairwise disjoint intervals in  $C_b$ , it is either a witness or a critical interval. If there do not exist  $k + 1$  pairwise disjoint intervals in  $C_b$  whose left endpoints are contained in  $R$ , then  $R$  is a *non-critical interval*. Consider the example in Figure 1(a). If  $k = 2$ ,  $R_1$  is a witness, since it contains  $B_1$ ,  $B_3$ , and  $B_4$ . If  $k = 3$ ,  $R_1$  is not a witness but is critical, since it contains the left endpoints of  $B_1$ ,  $B_3$ ,  $B_4$ , and  $B_5$ . For  $k \geq 4$ ,  $R_1$  is non-critical.

By Lemma 2, if  $C_r$  contains a witness,  $(C_r, C_b)$  is infeasible. Before proceeding, we remark that even if there is no witness in  $C_r$ , the feasibility of  $(C_r, C_b)$  is undetermined. Consider the example of  $C_r = \{[1, 4], [4, 7]\}$ ,  $C_b = \{[1, 1], [2, 2], [3, 5], [6, 6], [7, 7]\}$ , and  $k = 2$ . In this example, no red interval is a witness. However, it is easy to check that any subset  $S'$  that covers all the blue intervals hits either  $R_1$  or  $R_2$  three times.



**Fig. 1.** An illustration of Algorithm 1, in which  $k = 3$

For convenience, we call  $(C_r, C_b)$  a *bad instance* (of the MDH problem) if  $C_r$  contains a witness. If  $(C_r, C_b)$  is not a bad instance, we call it an *ambiguous instance* if it contains a critical interval. If every interval in  $C_r$  is neither a witness nor a critical interval, we call  $(C_r, C_b)$  a *good instance*.

Our algorithm consists of two phases. Phase 1 checks whether  $(C_r, C_b)$  is feasible. In addition, if  $(C_r, C_b)$  is feasible, Phase 1 reduces  $(C_r, C_b)$  into an equivalent good instance  $(C_r, C'_b)$ . Next, if  $(C_r, C_b)$  is feasible, Phase 2 further finds a feasible solution by solving the good instance  $(C_r, C'_b)$ .

By Lemma 2, any bad instance is infeasible. Given a good instance  $(C_r, C_b)$ , Phase 2 finds a feasible solution  $S'$  in  $O(n)$  time as follows: examine the intervals  $B_i$  from right to left and include  $\text{left}(B_i)$  into  $S'$  if no element of  $B_i$  is contained in the current  $S'$ . Our Phase 2 algorithm shows that any good instance is feasible.

### 3.2 Algorithm for Phase 1

Phase 1 examines each interval in  $C_r$ , from left to right, to see whether it is a witness. The checking of an interval  $R_j$  is described as follows. For each interval  $B_i$ , let  $\pi(B_i)$  be the first blue interval to its right that is disjoint from  $B_i$ . For each interval  $B_i$  and  $m \geq 1$ , define a function  $\pi^m(B_i)$  as follows:  $\pi^1(B_i) = \pi(B_i)$  and  $\pi^m(B_i) = \pi^{m-1}(\pi(B_i))$  for  $m \geq 2$ . For example, in Figure 1(a),  $\pi^3(B_1) = \pi^2(B_3) = \pi^1(B_4) = B_5$ . For ease of presentation, we assume that there are

infinite pseudo blue intervals to the right of  $B_{|C_b|}$  which are pairwise disjoint, so that  $\pi^m(B_i)$  is well-defined.

Let  $\tau(R_j)$  be the first blue interval  $B_i$  whose left endpoint is contained in  $R_j$ . Let  $\beta(R_j) = \pi^k(\tau(R_j))$ . For example, in Figure 1(a),  $\tau(R_1) = B_1$  and if  $k = 3$ ,  $\beta(R_1) = \pi^3(B_1) = B_5$ . Since  $C_b$  has the non-nested property and the intervals in  $C_b$  are ordered by increasing left endpoints,  $\text{left}(\beta(R_j))$  is the smallest integer  $x$  such that  $[\text{left}(R_j), x]$  contains the left endpoints of  $k + 1$  pairwise disjoint intervals in  $C_b$ , and  $\text{right}(\beta(R_j))$  is the smallest integer  $x$  such that  $[\text{left}(R_j), x]$  contains  $k + 1$  pairwise disjoint intervals in  $C_b$ . Therefore, whether  $R_j$  is a critical interval or a witness can be determined, respectively, by the following two procedures.

**Procedure** CRITICAL( $R_j, \beta(R_j)$ )

**begin**

1. **if**  $(\text{right}(R_j) \geq \text{left}(\beta(R_j)))$  and  $(\text{right}(R_j) < \text{right}(\beta(R_j)))$   
**then return** TRUE

2. **else return** FALSE

**end**

**Procedure** WITNESS( $R_j, \beta(R_j)$ )

**begin**

1. **if**  $\text{right}(R_j) \geq \text{right}(\beta(R_j))$  **then return** TRUE

2. **else return** FALSE

**end**

If  $\text{WITNESS}(R_j, \beta(R_j))$  returns FALES, we say that  $R_j$  passes the witness-checking. As mentioned, even all intervals  $R_j$  pass the witness-checking, the feasibility of  $(C_r, C_b)$  is undetermined. Our intent is to give an algorithm such that  $(C_r, C_b)$  is feasible if and only if all  $R_j$  pass the witness-checking. To achieve this goal, if an interval  $R_j$  passes the witness-checking, we further check whether it is a critical interval. If it is, we update  $C_b$  to make  $R_j$  a non-critical interval. More specifically, if  $R_j$  is a critical interval, we update  $C_b$  to produce an equivalent instance in which  $R_1, R_2, \dots, R_j$  are all non-critical. Consequently, once all  $R_j$  pass the witness-checking, the given instance  $(C_r, C_b)$  is reduced to an equivalent good instance and we can conclude that it is feasible. We proceed to describe the update of  $C_b$  for a critical interval  $R_j$ .

**Lemma 3.** If  $R_j$  is a critical interval in  $C_r$ , no feasible solution to  $(C_r, C_b)$  contains an element in  $[\text{left}(\beta(R_j)), \text{right}(R_j)]$ .

Given an interval  $B_i$  and an element  $c \in B_i$ , define  $\text{CUT}(B_i, c)$  to be an operation that removes  $[\text{left}(B_i), c]$  from  $B_i$  (i.e., update  $\text{left}(B_i)$  into  $c + 1$ ). Given an interval  $B_i$ , define  $\text{TRIM}(B_i)$  to be an operation that removes all intervals in  $C_b$  that contain  $B_i$ . When a critical interval  $R_j$  is encountered, we perform the following procedure to make  $R_j$  non-critical.

```

Procedure UPDATE( $R_j, \beta(R_j)$ )
begin
1. CUT( $\beta(R_j), right(R_j)$ )
2. TRIM( $\beta(R_j)$ )
end

```

Consider the interval  $R_1$  in Figure 1(a). In this example,  $k = 3$ ,  $R_1$  is critical, and  $\beta(R_1) = B_5$ . Thus, UPDATE( $R_1, B_5$ ) is performed, in which the CUT operation shortens  $B_5$  and then the TRIM operation removes the intervals  $B_6$  and  $B_7$ . (See Figure 1(b).) Note that after UPDATE( $R_j, \beta(R_j)$ ),  $C_b$  still has the non-nested property and the intervals in  $C_b$  are still in order of increasing left endpoints.

**Lemma 4.** If  $R_j$  is a critical interval, UPDATE( $R_j, \beta(R_j)$ ) makes  $R_j$  non-critical.

**Lemma 5.** If  $R_j$  is a critical interval, UPDATE( $R_j, \beta(R_j)$ ) reduces  $(C_r, C_b)$  into an equivalent instance.

Our algorithm for Phase 1 is as follows.

**Algorithm 1.** Phase 1

```

begin
1. for  $j \leftarrow 1$  to  $|C_r|$  do /* examine the red intervals from left to right
2. begin
3.    $\tau(R_j) \leftarrow$  the first interval  $B_i$  (in the current  $C_b$ ) with  $left(B_i) \geq left(R_j)$ 
4.    $\beta(R_j) \leftarrow \pi^k(\tau(R_j))$  (with respect to the current  $C_b$ )
5.   if WITNESS( $R_j, \beta(R_j)$ ) = TRUE
        then output FALSE /* the input is infeasible
6.   if CRITICAL( $R_j, \beta(R_j)$ ) = TRUE
        then UPDATE( $R_j, \beta(R_j)$ ) /* make  $R_j$  non-critical
7. end
8. output  $(C_r, C_b)$  /* the input is feasible
end

```

Figure 1 gives illustrations of Algorithm 1. The correctness of Algorithm 1 is discussed as follows. By Lemma 5, the UPDATE operation in line 6 always reduces  $(C_r, C_b)$  into an equivalent instance. Thus, if Algorithm 1 outputs FALSE, the given input is certainly infeasible. If all  $R_j$  pass the witness-checking in line 5, the following lemma shows that Algorithm 1 reduces the given input into a good instance, from which we conclude that the input is feasible.

**Lemma 6.** Suppose that all  $R_j$  pass the witness-checking in line 5 of Algorithm 1. After the  $j$ th iteration of the for-loop, the input is reduced to an instance in which  $R_1, R_2, \dots, R_j$  are non-critical, where  $1 \leq j \leq |C_r|$ .

**Theorem 1.** If the input  $(C_r, C_b)$  is infeasible, Algorithm 1 outputs FALSE; otherwise, it outputs a good instance that is equivalent to  $(C_r, C_b)$ .

The running time of Algorithm 1 is analyzed as follows. Consider a fixed iteration of the for-loop. The finding of  $\tau(R_j)$  in line 3 is done by a scan on the intervals in the current  $C_b$ , from left to right. The scan starts from  $\tau(R_{j-1})$  and stops when  $\tau(R_j)$  is encountered. Thus, line 3 takes  $O(1)$  amortized time. Given  $\tau(R_j)$ , the finding of  $\beta(R_j)$  in line 4 can be easily done in  $O(|C_b|)$  time. Line 5 takes  $O(1)$  time. If  $R_j$  is critical, the call to UPDATE in line 6 removes a set  $\Delta_j$  of intervals from  $C_b$ . It is easy to check that  $\Delta_j$  is the set of intervals in  $C_b$  with left endpoints contained in  $(\text{left}(\beta(R_j)), \text{right}(R_j) + 1]$ . Since the intervals in  $C_b$  are ordered by increasing left endpoints, given the interval  $\beta(R_j)$ , the removal of  $\Delta_j$  can be easily done in  $O(|\Delta_j|)$  time. Therefore, the overall time complexity of Algorithm 1 is  $O(|C_r|n + \sum |\Delta_j|) = O(|C_r|n + |C_b|) = O(n^2)$ .

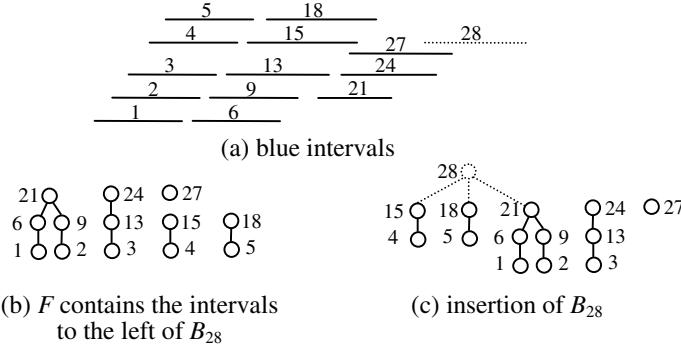
## 4 An $O(n\alpha(n))$ -Time Algorithm

The bottleneck of the algorithm in Section 3 is the finding of  $\beta(R_j)$  in line 4 of Algorithm 1. In this section, we show that the finding can be done in amortized  $O(\alpha(n))$  time. Our idea is to maintain a dynamic forest  $F$  for the intervals in  $C_b$  according to the  $\pi$  function, so that the finding of  $\beta(R_j)$  can be done by an upward traversal from a node in  $F$ . More specifically, each node in  $F$  represents an interval  $B_i$  and the parent of an interval  $B_i$  is  $\pi(B_i)$ . Then,  $\beta(R_j)$  is found by searching the  $(k+1)$ th node on the path from  $\tau(R_j)$  to its root.

If  $C_b$  is static, it is not difficult to construct the forest  $F$  in linear time. However, since  $C_b$  may be modified by UPDATE, we need to maintain  $F$  dynamically. For ease of discussion, in the remainder of this section, we assume that Algorithm 1 does not stop in any iteration of  $j < |C_r|$ . Let  $C_b^j$  denote the content of  $C_b$  at the end of the  $j$ th iteration. When  $C_b$  is modified by UPDATE, we may need to delete some nodes in  $F$  and link their children to new parents. To avoid deletion of nodes, we do not maintain all intervals of  $C_b$  in  $F$ . Instead, at the end of the  $j$ th iteration of Algorithm 1,  $F$  maintains only the intervals in  $C_b^j$  that are to the left of  $\beta(R_j)$ . Figure 2(a)(b) gives an illustration, in which  $\beta(R_j) = B_{28}$ .

Given two roots  $t$  and  $t'$  in  $F$ , define  $\text{LINK}(t, t')$  to be an operation that makes  $t'$  the rightmost child of  $t$ . Given a node  $v$  in  $F$ , define  $\text{FINDROOT}(v)$  to be an operation that returns a pair  $(r, d)$ , where  $r$  is the root of  $v$  in  $F$  and  $d$  is the number of edges from  $v$  to  $r$ . Alstrup and Holm's [2] had an efficient algorithm for the level-ancestor problem in a dynamic tree. With some modification, their algorithm can be used to support each  $\text{LINK}$  and  $\text{FINDROOT}$  operation in amortized  $O(\alpha(n))$  time [1].

Initially,  $F$  is empty. In the  $j$ th iteration of Algorithm 1, we insert to  $F$  those intervals in  $C_b^j$  which are to the left of  $\beta(R_j)$  but not in  $F$ . For ease of description, we call  $i$  the *index* of  $B_i$ . The index of  $\beta(R_j)$  is non-decreasing as  $j$  increases. Since  $\text{UPDATE}(R_j, \beta(R_j))$  does not remove or modify any interval to the left of  $\beta(R_j)$ , it is easy to conclude the following.

**Fig. 2.** An illustration of the forest  $F$ 

**Lemma 7.** If an interval  $B_i$  is inserted to  $F$  in the  $j$ th iteration, it will not be deleted from  $C_b$  in the remaining iterations.

We proceed to present the detailed maintenance of  $F$ . Consider the  $j$ th iteration of Algorithm 1. At the beginning, the set of intervals in  $F$  is the set of intervals in  $C_b^{j-1}$  that are to the left of  $\beta(R_{j-1})$ . The following procedure computes  $\beta(R_j)$  and updates  $F$ .

**Procedure COMPUTING- $\beta$**

**begin**

1.  $p \leftarrow$  the index of  $\beta(R_{j-1})$
  2. **if**  $\tau(R_j) \in F$  **then**  $(B_t, k') \leftarrow \text{FINDROOT}(\tau(R_j))$
  3.     **else**  $(B_t, k') \leftarrow (\tau(R_j), 0)$
  4.  $B_{p'} \leftarrow \pi^{k-k'}(B_t)$  /\* note that  $B_{p'} = \pi^{k-k'}(\pi^{k'}(\tau(R_j))) = \beta(R_j)$
  5. **for**  $i = p$  to  $p' - 1$  **do**
  6.     **if**  $B_i$  exists in the current  $C_b$  **then** insert  $B_i$  into  $F$
  7. **return**  $(B_{p'})$
- end**

Line 1 takes  $O(1)$  time. Lines 2-3 computes  $B_t = \pi^{k'}(\tau(R_j))$  in amortized  $O(\alpha(n))$  time, where  $\pi^0(\tau(R_j)) = \tau(R_j)$ . Note that  $k' < k$ , since the index of  $\beta(R_j)$  is not smaller than that of  $\beta(R_{j-1})$  and  $F$  maintains only intervals to the left of  $\beta(R_{j-1})$ . Line 4 is done in  $O(p' - p)$  time by a left-to-right scan on the intervals in the current  $C_b$ , starting from  $B_p$  and stopping when  $\pi^{k-k'}(B_t)$  is encountered.

We proceed to discuss the insertion of  $B_i$  into  $F$  in line 9. (See Figure 2(b).) Before the insertion, the set of intervals in  $F$  is the set of intervals in  $C_b^j$  that are to the left of  $B_i$ . Thus, our job is to create a root node  $B_i$  and to make each  $B_{i'}$  with  $\pi(B_{i'}) = B_i$  a child of  $B_i$  by invoking  $\text{LINK}(B_i, B_{i'})$ . Let  $Q$  be the set of intervals that should be linked to  $B_i$ . Since  $\text{left}(B_i)$  is larger than the left endpoint of any interval currently in  $F$ , the occurrence of  $B_i$  does not change any edge in the current  $F$ . Thus,  $Q$  is a subset of the roots. For any  $m \geq 1$ , the

index of  $\pi^m(B_i)$  is non-decreasing as  $i$  increases. Thus, we have the following, which is useful to the identification of  $Q$ .

**Lemma 8.** Let  $m \geq 1$  be an integer. Let  $B_x$  and  $B_y$  be two intervals in  $C_b^j$  such that  $x < y$  and  $\pi^m(B_x) = \pi^m(B_y)$ . Then,  $\pi^m(B_z)$  is the same for all intervals  $B_z$  with  $x \leq z \leq y$  in  $C_b^j$ .

From Lemma 8, we conclude that the roots in  $F$  are those intervals that have the first  $g$  largest indices (among the intervals in  $F$ ), where  $g$  is the number of roots. Moreover, the set  $Q$  is the set of roots that have the first  $|Q|$  smallest indices. Therefore,  $Q$  can be found by a simple scan on the roots in  $F$ , in order of increasing indices. The scan stops when a root intersecting  $B_i$  is encountered or all roots are examined. Thus, the computation of  $Q$  takes  $O(|Q|)$  time. And therefore, COMPUTING- $\beta$  requires  $O(\alpha(n) + (p' - p) + \delta_j \alpha(n))$  amortized time, where  $\delta_j$  is the total number of intervals and edges added to  $F$  at the  $j$ th iteration. Since each interval  $B_i$  is inserted into  $F$  at most once,  $O(\sum \delta_i) = O(n)$ . Consequently, the time complexity of the algorithm in Section 3.2 is reduced to  $O(n\alpha(n))$ .

## 5 A Linear-Time Algorithm

In Section 4, a forest  $F$  is used to help the finding of each  $\beta(R_j)$ . By using the algorithm in [2], each  $\beta(R_j)$  is found in amortized  $O(\alpha(n))$  time. Instead of using the algorithm in [2], this section explores more properties of  $F$  and presents a very simple data structure that supports the finding of each  $\beta(R_j)$  in amortized  $O(1)$  time. We assume that the children of each node in  $F$  are ordered by increasing indices. Consider a tree with root  $t$  in  $F$ . For  $m \geq 0$ , let  $V(t, m)$  be the set of nodes at depth  $m$ . By definition, each node  $v$  in  $V(t, m)$  has  $\pi^m(v) = t$ . Recall that intervals are inserted to  $F$  in order of increasing index. The  $a$ th interval inserted into  $F$  is associated with a *rank*  $a$ . The rank of an interval  $B_i$  increases as  $i$  increases. Therefore, we can obtain the following from Lemma 8.

**Lemma 9.** For any root  $t$  in  $F$  and depth  $m \geq 0$ , the nodes in  $V(t, m)$  have consecutive ranks.

By Lemma 9,  $V(t, m)$  can be represented by an interval of ranks, denoted by  $A(t, m)$ . To speedup the finding of each  $\beta(R_j)$ , more information is maintained. Let  $T$  be a tree with root  $t$  in  $F$ . The root  $t$  is associated with an integer  $h(t)$  and two intervals  $L(t)$  and  $N(t)$ , where  $h(t)$  records the height of  $T$ ,  $L(t)$  records  $A(t, h(t))$ , which represents the leaves of  $T$ , and  $N(t)$  records  $A(t, k - 1)$ , which represents the nodes at depth  $k - 1$ .

Each node  $v$  in  $T$  is associated with two pointers  $p_{next}(v)$  and  $p_{last}(v)$ . Consider the nodes in  $V(t, m)$  for a fixed depth  $m < h(t)$ . Assume that the nodes are in order of increasing ranks. Let  $B_a$  and  $B_b$  be, respectively, the first and last nodes in  $V(t, m)$ . Let  $B_x$  and  $B_y$  be, respectively, the first and last nodes

in  $V(t, m)$  that are internal nodes of  $T$ . We say that the left side of  $V(t, m)$  is *ready* if either (i)  $B_x = B_a$  and  $B_x$  points to itself by using  $p_{last}(B_x)$ , or (ii)  $B_x \neq B_a$  and  $B_a$  and  $B_x$  point to each other by using  $p_{next}(B_a)$  and  $p_{last}(B_x)$ . Similarly, we say that the right side of  $V(t, m)$  is *ready* if either (i)  $B_y = B_b$  and  $B_y$  points to itself by using  $p_{next}(B_y)$ , or (ii)  $B_y \neq B_b$  and  $B_y$  and  $B_b$  point to each other by using  $p_{next}(B_y)$  and  $p_{last}(B_b)$ . We say that  $V(t, m)$  is *ready* if its both sides are ready. The pointers  $p_{next}$  and  $p_{last}$  are maintained such that  $V(t, m)$  is ready for any depth  $m < h(t)$ .

**Lemma 10.** Given a root  $t$  in  $F$ , the interval  $A(t, k - 2)$  can be determined in  $O(1)$  time.

We proceed to describe the insertion of an interval  $B_i$  into  $F$ . Recall that  $\text{LINK}(t, t')$  is an operation that makes  $t'$  the rightmost child of  $t$ . The insertion of  $B_i$  is done as follows. Let  $g$  be the rank of  $B_i$  and let  $Q$  be the set of roots in  $F$  that should be linked to  $B_i$ . First, we create a tree of a single node to represent  $B_i$ , in which  $h(B_i) = 0$ ,  $L(B_i) = [g, g]$ ,  $N(B_i)$  is  $\emptyset$  if  $k > 1$  and is  $[g, g]$  otherwise, and  $p_{last}(B_i) = p_{next}(B_i) = B_i$ . Next, repeatedly, we make each root  $t'$  in  $Q$ , in order of increasing index, a child of  $B_i$  by invoking  $\text{LINK}(B_i, t')$ .

The implementation of  $\text{LINK}(t, t')$  is discussed as follows. If we make  $t'$  the rightmost child of  $t$ ,  $V(t', m - 1)$  is included to  $V(t, m)$  for  $1 \leq m \leq h(t') + 1$ . Thus, by Lemma 9, the nodes in  $V(t, m)$  and  $V(t', m - 1)$  have consecutive ranks. More specifically, if both  $V(t, m)$  and  $V(t', m - 1)$  are non-empty, we have  $\text{right}(A(t, m)) = \text{left}(A(t', m - 1)) - 1$ . Let  $h = h(t)$  and  $h' = h(t')$ . Three cases are considered.

**Case 1:  $h = h' + 1$ .** The height  $h(t)$  remains the same. We update  $L(t)$  to  $L(t) \cup L(t')$ , which is  $[\text{left}(L(t)), \text{right}(L(t'))]$ , and update  $N(t)$  to  $N(t) \cup A(t', k - 2)$ , which is  $[\text{left}(N(t)), \text{right}(A(t', k - 2))]$ . By Lemma 10,  $A(t', k - 2)$  is found in  $O(1)$  time. The problem remains is to update some pointers  $p_{next}$  and  $p_{last}$ , so that the new  $V(t, m)$  is ready for each  $m < h$ . Since  $V(t, 0)$  remains the same, update is not required for  $m = 0$ . Consider a fixed  $m$  with  $1 \leq m < h$ . Before  $\text{LINK}(t, t')$ ,  $V(t, m)$  and  $V(t', m - 1)$  are both ready. Note that since  $m < h$ , each of  $V(t, m)$  and  $V(t', m - 1)$  contains at least one internal node. After  $\text{LINK}(t, t')$ , the first node and the first internal node in  $V(t, m)$  remain the same; and the last node and last internal node in  $V(t', m - 1)$  become those of the new  $V(t, m)$ . Consequently, after including  $V(t', m - 1)$ ,  $V(t, m)$  is still ready. Thus, for  $1 \leq m < h$ , update is also not required.

**Case 2:  $h > h' + 1$ .** The height  $h(t)$  and the interval  $L(t)$  remain the same. We update  $N(t)$  to  $N(t) \cup A(t', k - 2)$ . Consider the update of  $p_{next}$  and  $p_{last}$  for each depth  $m$ , where  $0 \leq m < h$ . Similar to Case 1, no update is required for  $0 \leq m \leq h'$ . Also, no update is required for  $m > h' + 1$ , since  $V(t, m)$  remains the same. Finally, consider the update for  $m = h' + 1$ . The left side of the original  $V(t, h' + 1)$  is ready. Thus, after  $\text{LINK}(t, t')$ , its left side is still ready. All nodes in  $V(t', h')$  are leaves. To make the right side of the new  $V(t, h' + 1)$  ready, we need to find the last node  $B_{b'}$  in  $V(t', h')$  and

the last internal node  $B_y$  in the original  $V(t, h' + 1)$  and then to make them point to each other. The node  $B_{b'}$  is the node whose rank is  $\text{right}(L(t'))$ . By Lemma 9, the last node in  $V(t, h' + 1)$ , denoted by  $B_b$ , is the node whose rank is  $\text{left}(L(t')) - 1$ . If  $B_b$  is an internal node, we have  $B_y = B_b$ ; otherwise, we have  $B_y = p_{\text{last}}(B_b)$ . Thus,  $B_y$  and  $B_{b'}$  can be found in  $O(1)$  time. And therefore, the update for  $m = h' + 1$  takes constant time.

**Case 3:  $h < h' + 1$ .** We update  $h(t)$  to  $h' + 1$ ,  $N(t)$  to  $N(t) \cup A(t', k - 2)$ , and  $L(t)$  to  $L(t')$ . The update of  $p_{\text{next}}$  and  $p_{\text{last}}$  is similar to Case 2.

Next, we proceed to present an algorithm for the finding of each  $\beta(R_j)$ . An extra pointer  $p$  is maintained. Initially,  $p$  points to  $B_1$ . After the  $j$ th iteration of Algorithm 1,  $p$  points to  $\pi^{k-1}(\tau(R_j))$ . Note that since  $F$  contains only intervals to the left of  $\beta(R_j)$ ,  $p$  is a root. We compute  $\beta(R_j)$  as follows.

**Step 1.** If  $\tau(R_j)$  is not an interval in the current  $F$ , adds un-inserted intervals in  $C_b^{j-1}$  into  $F$ , in increasing order of index, until  $\tau(R_j)$  becomes one. Let  $x$  be the rank of  $\tau(R_j)$ .

**Step 2.** Scan the roots in  $F$ , starting from  $p = \pi^{k-1}(\tau(R_{j-1}))$ , in increase order of index, until a root  $t$  with  $x \in N(t)$  (i.e.,  $t = \pi^{k-1}(\tau(R_j))$ ) is encountered. If no such  $t$  exists, adds un-inserted intervals in  $C_b^{j-1}$  into  $F$ , in increasing order of index, until  $t$  appears.

**Step 3.** Find the interval  $\pi(t)$ , which is obtained by examining the un-inserted intervals in  $C_b^{j-1}$ , in increasing order of index, until an interval that is disjoint from  $t$  appears.

**Step 4.** Set  $\beta(R_j) = \pi(t)$  and  $p = t$ .

Each LINK operation takes  $O(1)$  time. Thus, each insertion to  $F$  requires  $O(1)$  amortized time. Consequently, we obtain the following.

**Theorem 2.** After an  $O(|C_r| + |C_b| + |S|)$ -time preprocessing, the MDH problem with  $C_r \cup C_b$  having the C1P can be solved in  $O(|S|)$  time.

## References

1. Alstrup, S.: Private communication (2012)
2. Alstrup, S., Holm, J.: Improved algorithms for finding level ancestors in dynamic trees. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 73–84. Springer, Heidelberg (2000)
3. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. Journal of Computer and System Sciences 13(3), 335–379 (1976)
4. Caprara, A., Toth, P., Fischetti, M.: Algorithms for the set covering problem. Annals of Operations Research 98, 353–371 (2000)
5. Carr, R.D., Doddi, S., Konjevod, G., Marathe, M.: On the red-blue set cover problem. In: Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 345–353 (2000)

6. Chang, M.S., Chung, H.H., Lin, C.C.: An improved algorithm for the red-blue hitting set problem with the consecutive ones property. *Information Processing Letters* 110(20), 845–848 (2010)
7. Dom, M., Guo, J., Niedermeier, R., Wernicke, S.: Red-blue covering problems and the consecutive ones property. *Journal of Discrete Algorithms* 6(3), 393–407 (2008)
8. Feder, T., Motwani, R., Zhu, A.: k-connected spanning subgraphs of low degree. *Tech. Rep. TR06-041, Electronic Colloquium on Computational Complexity* (2006)
9. Kuhn, F., von Rickenbach, P., Wattenhofer, R., Welzl, E., Zollinger, A.: Interference in cellular networks: The minimum membership set cover problem. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 188–198. Springer, Heidelberg (2005)
10. Mecke, S., Schöbel, A., Wagner, D.: Station location - complexity and approximation. In: 5th Workshop on Algorithmic Methods and Models for Optimization of Railways (2005)
11. Mecke, S., Wagner, D.: Solving geometric covering problems by data reduction. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 760–771. Springer, Heidelberg (2004)
12. Ruf, N., Schobel, A.: Set covering with almost consecutive ones property. *Discrete Optimization* 1(2), 215–228 (2004)
13. Veinott, A.F., Wagner, H.M.: Optimal capacity scheduling. *Operations Research* 10(4), 518–532 (1962)
14. Wang, B.F., Li, C.H.: On the minimum degree hypergraph problem with subset size two and the red-blue set cover problem with the consecutive ones property. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 169–180. Springer, Heidelberg (2012)

# On the Conjunctive Capacity of Graphs

Miroslav Chlebík<sup>1</sup> and Janka Chlebíková<sup>2</sup>

<sup>1</sup> University of Sussex, UK

[m.chlebik@sussex.ac.uk](mailto:m.chlebik@sussex.ac.uk)

<sup>2</sup> University of Portsmouth, UK

[janka.chlebikova@port.ac.uk](mailto:janka.chlebikova@port.ac.uk)

**Abstract.** The investigation of the asymptotic behaviour of various graph parameters in powers of a fixed graph  $G = (V, E)$  is motivated by problems in information theory and extremal combinatorics. Considering various parameters and/or various notions of graph powers we can arrive at different notions of graph capacities, of which the Shannon capacity is best known. Here we study a related notion of the so-called conjunctive capacity of a graph  $G$ ,  $C_{\text{AND}}(G)$ , introduced and studied by Gargano, Körner and Vaccaro [5], [6]. To determine  $C_{\text{AND}}(G)$  is a convex programming problem. In this paper we show that the optimal solution to this problem is unique and describe the structure of the solution in any (simple) graph. We show that its reciprocal value  $vc_C(G) := \frac{1}{C_{\text{AND}}(G)}$  is an optimal solution of the newly introduced problem of Minimum Capacitary Vertex Cover that is closely related to the LP-relaxation of the Minimum Vertex Cover Problem. We also describe its close connection with the binding number/binding set of a graph, and with the strong crown decomposition of graphs introduced in [2].

**Keywords:** graph capacities, compound channel, Shannon capacity for graph families, fractional vertex cover, binding number, strong crown decomposition.

## 1 Introduction

An induced complete subgraph of a graph  $G$  is called a clique and the cardinality of the largest clique of  $G$  is called its clique number,  $\omega(G)$ . The analysis of its growth in large product graphs leads to several interesting problems in combinatorics. The problem was originated by Shannon [11] in 1956 in his analysis of the capability of certain noisy communication channels to transmit information in an error-free manner. Shannon's model associated a graph with every channel. In our notation the vertex set of the graph represents the symbols that can be transmitted through the channel and two vertices are connected by an edge if the corresponding symbols can never get confused by the receiver. This model naturally leads to a product of graphs through the repeated use of the channel for the transmission of symbol sequences of some fixed length  $n$ . If the graph  $G = (V, E)$  is a simple graph (all graphs in this paper are assumed simple), then  $G^n$  denotes the graph with vertex set  $V^n$  whose edge set contains those

pairs of sequences in  $V^n$  which can never get confused by the receiver. Formally,  $\{x, y\} \in E(G^n)$  if and only if  $\exists i \{x_i, y_i\} \in E$ , where  $E$  denotes the edge set of the graph  $G$  and  $x = (x_1, x_2, \dots, x_n)$ ,  $y = (y_1, y_2, \dots, y_n)$  are elements of  $V^n$ . Following Berge [1],  $G^n$  is called the  $n$ -th co-normal power of  $G$ . Shannon [11] observed that if  $K$  is a clique in  $G$  then  $K^n$  is a clique in  $G^n$ , whence the size of the clique number of  $G^n$  is at least the  $n$ -th power of the clique number of  $G$ . (In fact, these two quantities coincide whenever the clique number of  $G$  equals its chromatic number. This observation led Berge to his celebrated concept of perfect graphs.) The observation logically leads to the concept introduced by Shannon – to determine the always existing limit  $C(G) = \lim_{n \rightarrow \infty} \frac{\log \omega(G^n)}{n}$ , which is called the Shannon capacity of  $G$ . An analogous approach was initiated in [9] where the notion of Sperner capacity was introduced as the natural counterpart of Shannon capacity in the case of directed graphs. These notions became the key to the solution of some important open problems in extremal combinatorics [5], [6].

We have to warn that many traditional papers use a complementary language and define Shannon capacity  $C(G)$  as our  $C(\overline{G})$ , when  $C(G)$  is then defined using independent sets in the normal powers instead of cliques in the co-normal powers of  $G$ .

One can also extend the definition of Shannon capacity to graph families. Let  $\mathcal{G} = \{G_1, G_2, \dots, G_k\}$  be a family of graphs on a common set of vertices  $V(G_i) = V$ . We define the Shannon capacity  $C(\mathcal{G})$  of a family  $\mathcal{G}$  by

$$C(\mathcal{G}) = \lim_{n \rightarrow \infty} \frac{1}{n} \log \omega(G_1^n \cap G_2^n \cap \dots \cap G_k^n).$$

The Shannon capacity of a family is motivated as the zero-error capacity of the compound channel described by  $\mathcal{G}$ . Additional motivation comes from extremal combinatorics.

Let us be given a graph  $G = (V, E)$ . The Shannon capacity can be considered an “OR-capacity” for  $G$ . In its definition, two elements  $x$  and  $y$  of  $V^n$  are considered “really different” if there is a coordinate  $i$  for which  $\{x_i, y_i\} \in E$ , i.e. *at least one* of the edges of  $G$  occurs among the coordinate pairs  $\{x_i, y_i\}$ . The important fact is that it can be *either one* of the edges of  $G$  – this is what we mean by calling  $C(G)$  above as OR-capacity.

The second natural capacity associated with the graph  $G$  is “AND-capacity” (the conjunctive capacity), denoted by  $C_{\text{AND}}(G)$ . In its definition one would require that *every edge* of the graph  $G$  be present among the coordinate pairs of the sequences. More explicitly, now we define the  $n$ -th conjunctive power of  $G$  as the graph  $G_{\text{AND}}^n = (V^n, E(G_{\text{AND}}^n))$  such that  $\{(x_1, x_2, \dots, x_n), (y_1, y_2, \dots, y_n)\} \in E(G_{\text{AND}}^n)$  if and only if for every  $e \in E$  there exists a coordinate  $1 \leq i \leq n$  such that  $\{x_i, y_i\} = e$ , and  $C_{\text{AND}}(G) = \lim_{n \rightarrow \infty} \frac{\log \omega(G_{\text{AND}}^n)}{n}$ .

The conjunctive capacity is clearly a special case of the Shannon capacity of a family of graphs. Given a graph  $G = (V, E)$ , let us denote by  $\mathcal{G} := \mathcal{F}(G)$  the family of single-edge graphs obtained by considering for every  $e \in E$  the graph  $G_e$  defined by setting  $V(G_e) = V$  and  $E(G_e) = \{e\}$ . Thus  $\mathcal{G} := \mathcal{F}(G)$  is

the family of the  $|E|$  different single-edge graphs and the conjunctive capacity,  $C_{\text{AND}}(G)$ , of  $G$  is just the Shannon capacity  $C(\mathcal{F}(G))$  of this simple family of graphs.

The problem of determining the Shannon capacity  $C(G)$  of a given graph  $G$  is not even known to be NP-hard although it seems plausible that it is in fact much harder; the problem of deciding whether the Shannon capacity of a given graph exceeds a given value is not known to be decidable. On the other hand, to determine the conjunctive capacity  $C_{\text{AND}}(G)$  is algorithmically much simpler.

In this paper we show that the reciprocal value  $\frac{1}{C_{\text{AND}}(G)}$  is an optimal solution of the newly introduced problem of Minimum Capacitary Vertex Cover that is closely related to the LP-relaxation of the Minimum Vertex Cover Problem (Section 2). We prove that the optimal solution to the problem is unique (Section 5) and describe the structure of the solution in any (simple) graph (Section 6). We also point out its close connection with the binding number/binding set of a graph and the strong crown decompositions of graphs described in [2].

## 2 The Conjunctive Capacity

In [3] and [5] the authors study the asymptotic value  $C_{\text{AND}}(G)$ . These results have been used to answer a long-standing open question on the asymptotics of the maximum number of qualitatively independent partitions in the sense of Rényi [10]. They provide a computable formula for determining the conjunctive capacity  $C_{\text{AND}}(G)$  of any graph  $G = (V, E)$  as

$$C_{\text{AND}}(G) = \max_P \min_{\{u, v\} \in E} (P(u) + P(v)) h\left(\frac{P(u)}{P(u) + P(v)}\right), \quad (1)$$

where the maximum is over all probability distributions  $P$  on the vertex set  $V$  of  $G$ ,  $h$  is the binary entropy function,  $h(t) = -t \log t - (1-t) \log(1-t)$ ,  $t \in (0, 1)$  which is continuously extended by  $h(0) = h(1) = 0$  to the interval  $[0, 1]$ . (Here and in the sequel logarithms are to the base 2.) The formula (1) is starting point to our results.

In this paper we study the structure of the optimal solution of the convex programming problem defined on the graph  $G$ , introduced by the right hand side of (1). In what follows we use the function  $f: [0, \infty) \times [0, \infty) \rightarrow [0, \infty)$  defined by  $f(x, y) = (x + y)h\left(\frac{x}{x+y}\right)$ ,  $(x, y) \in [0, \infty) \times [0, \infty) \setminus \{(0, 0)\}$ , and  $f(0, 0) = 0$ . Notice that  $f$  is continuous on  $[0, \infty) \times [0, \infty)$ , and it simplifies to  $f(x, y) = (x + y) \log(x + y) - x \log x - y \log y$  for  $x, y > 0$ .

Let  $G = (V, E)$  be a graph. The conjunctive capacity of  $G$  can then be expressed as

$$C_{\text{AND}}(G) = \max_P \min_{\{u, v\} \in E} f(P(u), P(v)) \quad (2)$$

where maximum is taken over all probability distributions  $P$  on  $V$ . (If  $G$  does not have any edges then we set  $C_{\text{AND}}(G) = \infty$ .) If  $G$  has at least one edge, a distribution  $P$  for which the maximum in the definition of  $C_{\text{AND}}(G)$  is achieved clearly exists. We will show later that such a distribution is unique and describe the structure of this optimal distribution.

### The Minimum Capacitary Vertex Cover Problem

Recall that in the LP-relaxation of Minimum Vertex Cover the task is for a given graph  $G = (V, E)$  to minimize its fractional vertex cover

$$vc_*(G) = \min_x \{x(V) : (x(u), x(v)) \in T \text{ for every } \{u, v\} \in E\}, \quad (3)$$

where  $T = \{(a, b) \in [0, \infty) \times [0, \infty) : a + b \geq 1\}$  and the minimum is taken over all  $x: V \rightarrow [0, \infty)$ . (Here and in the sequel  $x(V) := \sum_{u \in V} x(u)$ .)

Now let us consider for a graph  $G = (V, E)$  and any fixed  $t > 0$  the following minimization problem over all nonnegative functions  $x: V \rightarrow [0, \infty)$ :

$$g(t) = \min_x \{x(V) : f(x(u), x(v)) \geq t \text{ for every } \{u, v\} \in E\}. \quad (4)$$

The edge constraints in (4) are similar to those of the LP-relaxation of the Minimum Vertex Cover problem for  $G$ . If we denote

$$T(t) = \{(a, b) \in [0, \infty) \times [0, \infty) : f(a, b) \geq t\}, \quad (5)$$

then (4) reads as  $g(t) = \min_x \{x(V) : (x(u), x(v)) \in T(t) \text{ for every } \{u, v\} \in E\}$ .

It can easily be verified by direct computation that the function  $f$  is positive homogeneous which means  $f(cx, cy) = cf(x, y)$  for each  $c > 0$  and each  $(x, y) \in [0, \infty) \times [0, \infty)$ . So the function  $g(t)$  has nice scaling properties, in particular  $g(t) = tg(1)$  for every  $t > 0$ . Hence  $C_{\text{AND}}(G)$  is the only  $t > 0$  such that  $g(t) = 1$  and consequently,  $g(1) = \frac{1}{C_{\text{AND}}(G)}$  as  $g(C_{\text{AND}}(G)) = 1$ .

To determine  $C_{\text{AND}}(G)$  and a maximizing distribution  $P$  for  $G$  in (2) we introduce the following optimization problem that we call the **Minimum Capacitary Vertex Cover** problem:

*Instance:* A graph  $G = (V, E)$ .

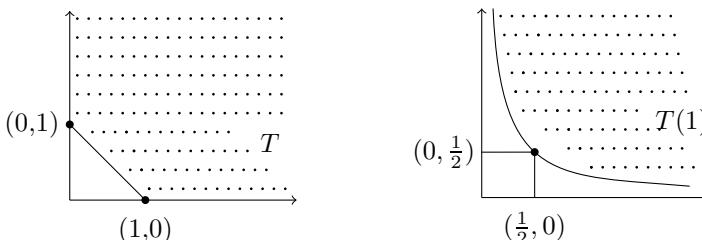
*Feasible solution:* A function  $x: V \rightarrow [0, +\infty)$  satisfying constraints  $(x(u), x(v)) \in T(1)$  for every edge  $\{u, v\} \in E$ , where  $T(1)$  is defined by

$$T(1) = \{(a, b) \in [0, \infty) \times [0, \infty) : f(a, b) \geq 1\}.$$

*Goal:* To minimize  $x(V) := \sum_{u \in V} x(u)$  over all feasible solutions.

Denote  $vc_C(G)$  the optimum value of the Minimum Capacitary Vertex Cover problem.

One can compare the shapes of  $T$  and  $T(1)$  (**Figure 1**):



The main difference is in the presence of flat parts of the boundary of  $T$  that is not present when dealing with  $T(1)$ . Except that the structure of the optimum solutions is similar for both problems as we will see later.

Clearly,  $vc_C(G) = \frac{1}{C_{\text{AND}}(G)}$ , and a minimizer  $x: V \rightarrow [0, \infty)$  realizing  $vc_C(G)$  corresponds to a scaled capacity distribution  $P$  that realizes  $C_{\text{AND}}(G)$ . In what follows we will deal with the functional  $vc_C(\cdot)$  rather than with  $C_{\text{AND}}(\cdot)$ , as  $vc_C(\cdot)$  appears to be additive with respect to certain natural decompositions of  $G$ .

### 3 Strong Crown Decomposition

In this section we recall some notions and results from our paper [2] trying to keep notation the same. Consider a graph  $G = (V, E)$ . For  $U \subseteq V$ , let  $N(U)$  denote the set of its neighbors in  $G$ ,  $N(U) := \{v \in V : \exists u \in U \text{ such that } \{u, v\} \in E\}$ , and  $G[U]$  be the subgraph of  $G$  induced by  $U$ .

Recall that a **strong crown** in a graph  $G = (V, E)$  is a nonempty independent set  $I$  of  $G$  such that  $|N(U) \cap I| > |U|$  holds for every nonempty set  $U \subseteq N(I)$ . If  $I$  is a strong crown in  $G$  then  $I$  is the only maximum independent set in  $G[I \cup N(I)]$  or even in the bipartite graph  $G[I, N(I)]$  obtained from  $G[I \cup N(I)]$  removing all edges within  $N(I)$  (if any). It turns out that graphs containing a strong crown can be recognized efficiently and its strong crown can be found efficiently. As it may not be unique our aim is to find a maximal one.

For any graph  $G = (V, E)$  there is a unique strong crown decomposition  $(I, H, K)$ , where  $I$  is a strong crown,  $H = N(I)$  and  $K = V \setminus (I \cup H)$  is such that  $G[K]$  contains no strong crowns. We will refer to  $(I, H, K)$  as the **canonical strong crown decomposition of  $G$**  in what follows (see [2] for more details).

We have  $I = \emptyset$  in the decomposition above exactly when  $(I, H, K) = (\emptyset, \emptyset, V)$ , which is equivalent to the graph  $G = (V, E)$  being Hallian; this means it satisfies Hall's property  $|N(I)| \geq |I|$  for each independent set  $I$  of  $G$  or, equivalently,  $|N(U)| \geq |U|$  for each  $U \subseteq V$ . In particular, the graphs  $G[K]$  obtained by this decomposition always satisfy the Hall's property.

In questions of conjunctive capacity studied in this paper, Hallian graphs appear to be trivial. Nontrivial graphs are those with a nontrivial strong crown part  $G[I \cup H]$  in their canonical strong crown decomposition.

### 4 Binding Number and Binding Set of a Graph

The concept of the binding number of a graph was introduced by Woodall [12] in 1973. The binding number of a graph  $G = (V, E)$ , denoted  $\text{bind}(G)$ , is given by

$$\text{bind}(G) = \min_{U \subseteq V} \left\{ \frac{|N(U)|}{|U|} : U \neq \emptyset, N(U) \neq V \right\}.$$

A **binding set of  $G$**  is any set  $U$  in  $G$  with  $\text{bind}(G) = \frac{|N(U)|}{|U|}$ .

There has been an increased interest in binding numbers as they may be related to other important graph properties. For example, if  $\text{bind}(G) \geq \frac{3}{2}$  and  $G$  has at least three vertices then  $G$  has a Hamiltonian circuit.

The binding number and the binding set can be computed in polynomial time ([4]). The approach to computing  $\text{bind}(G)$  is based on a standard idea

for ratio minimization. Let's consider the problem of minimizing the difference  $|N(U)| - \lambda|U|$ , for  $\lambda \geq 0$  a fixed number. If  $d(G, \lambda)$  denotes  $\min\{|N(U)| - \lambda|U| : U \subseteq V, U \neq \emptyset, N(U) \neq V\}$ , it is easy to see that  $bind(G) \geq \lambda$  if and only if  $d(G, \lambda) \geq 0$ .

Since  $bind(G)$  is a rational number whose numerator and denominator are bounded by  $|V|$ , we can deal with  $\lambda$  of this form only. The corresponding minimization problem can be solved by a network flow method, but the condition  $N(U) \neq V$  makes the problem difficult. We point out that in the situation of our main interest, namely,  $bind(G) < 1$ ,  $bind(G)$  can be computed faster with one minimum cut calculation in time  $O(\log |V|)$ . The reason behind this is that if  $\lambda < 1$ , the restriction that  $N(U) \neq V$  can be dropped from the definition of  $d(G, \lambda)$  without changing its value (if there is a binding set  $U$  with property  $N(U) = V$  then  $bind(G) \geq 1$ ).

It is more relevant to focus our attention to the problem of determining the truncated version of the binding number (and binding set) problem, namely  $\min\{bind(G), 1\}$ . Given a graph  $G$ , we either conclude that  $bind(G) \geq 1$  or, if  $bind(G) < 1$ , we want to find inclusionwise maximal binding set  $U$ . If  $bind(G) < 1$  then every binding set  $U$  of  $G$  is an independent set of  $G$  ([8]). In the following lemma we prove some important properties of binding sets.

**Lemma 1.** *Let  $G$  be a graph and let  $(I, H, K)$  be its canonical strong crown decomposition. If  $bind(G) < 1$  then*

- (i) *every binding set of  $G$  is contained in  $I$ ,*
- (ii) *whenever  $U$  and  $W$  are two binding sets of  $G$  then  $U \cup W$  is a binding set, and  $U \cap W$  is either empty or a binding set; in particular, the union of all binding sets of  $G$  is a binding set.*

*Remark 1.* Given any graph  $G = (V, E)$  with  $bind(G) < 1$ , it is an important partial problem to find the unique inclusionwise maximal binding set  $I^*$  of  $G$ . We know that we can compute efficiently a binding set  $I_1$ , let  $H_1 = N(I_1)$  and  $G_1 = G[V \setminus (I_1 \cup H_1)]$ . One can observe that  $bind(G_1) \geq bind(G)$  (otherwise for a binding set  $I_2$  of  $G_1$  setting  $H_2 = N_{G_1}(I_2)$  we would get  $\frac{|H_1 \cup H_2|}{|I_1 \cup I_2|} < bind(G)$ , a contradiction).

Moreover,  $bind(G_1) > bind(G)$  holds if and only if  $I_1 = I^*$ . Firstly, if  $bind(G_1) = bind(G)$  we get that  $\frac{|H_1 \cup H_2|}{|I_1 \cup I_2|} = bind(G)$ , so  $I_1 \cup I_2$  is a larger binding set of  $G$  and, in particular,  $I_1$  cannot be whole of  $I^*$ . Secondly, if  $I_1 \neq I^*$  then  $bind(G_1) = bind(G)$ . To show that, it is sufficient to show  $bind(G_1) \leq bind(G)$ . Setting  $H^* = N(I^*)$ ,  $b := bind(G)$ , we have  $|H^*| = b|I^*|$ ,  $|H_1| = b|I_1|$  and consequently  $N_{G_1}(I^* \setminus I_1) = H^* \setminus H_1$  and  $|H^* \setminus H_1| = b|I^* \setminus I_1|$ , showing  $bind(G_1) \leq bind(G)$ .

Now we can set  $G_2 = G[V \setminus (I_1 \cup I_2 \cup H_1 \cup H_2)]$ . If  $I_1 \cup I_2$  is not  $I^*$ , i.e.  $bind(G_2) = bind(G)$ , for a binding set  $I_3$  of  $G_2$  that we then compute with  $H_3 = N_{G_2}(I_3)$  we get that  $I_1 \cup I_2 \cup I_3$  is a larger binding set of  $G$ . We can continue in this way until  $I_1 \cup I_2 \cup \dots \cup I_k$  will be that maximal binding set  $I^*$  of  $G$ .

## 5 Minimum Capacitary Vertex Cover and Its Properties

Now we study the properties of the optimal solutions of the Minimum Capacitary Vertex Cover problem. Recall that this is a convex programming problem defined on a graph  $G = (V, E)$  by

$$vc_C(G) = \min_x \{x(V) : (x(u), x(v)) \in T(1) \text{ for every } \{u, v\} \in E\}.$$

Here  $T(1) = \{(a, b) \in [0, \infty) \times [0, \infty) : f(a, b) \geq 1\}$ , where  $f(x, y) = (x + y) \log(x + y) - x \log x - y \log y$  for  $x, y > 0$ , extended continuously to  $[0, \infty) \times [0, \infty)$ .

It is important that  $f$  is concave on  $(0, \infty) \times (0, \infty)$ . To verify this, we can compute the Hessian matrix  $H(x, y)$  of  $f(x, y)$  which reads as

$$H(x, y) = \begin{pmatrix} \frac{1}{x+y} - \frac{1}{x}, & \frac{1}{x+y} \\ \frac{1}{x+y}, & \frac{1}{x+y} - \frac{1}{y} \end{pmatrix} \text{ and so } \det(\lambda I - H(x, y)) = \lambda \left( \lambda + \frac{1}{x} + \frac{1}{y} - \frac{2}{x+y} \right).$$

As its eigenvalues are non-positive,  $H(x, y)$  is negative semidefinite, and so  $f(x, y)$  is concave. Consequently,  $T(1)$  is a convex set.

Let us denote by  $\varphi(x)$  the function whose graph describes the boundary  $\partial T(1)$ . For any  $x \in (0, \infty)$ ,  $\varphi(x)$  denotes a unique  $y = \varphi(x)$  such that  $(x + y) \log(x + y) - x \log x - y \log y = 1$ . As  $T(1)$  is symmetric with respect to the line  $y = x$ ,  $\varphi(\varphi(x)) = x$  for each  $x \in (0, \infty)$ . Moreover,  $\varphi(\frac{1}{2}) = \frac{1}{2}$ , and  $\varphi : (0, \infty) \rightarrow (0, \infty)$  is a real analytic function. We can observe that  $\partial T(1)$  does not contain any line segment, as the uniqueness theorem for real analytic functions would allow such behaviour of  $\varphi$  only for functions that are affine globally in  $(0, \infty)$ , which  $\varphi$  certainly is not. We can thus conclude that  $T(1)$  is strictly convex. The fact that its boundary does not contain any line segments will be important in our proof that the optimal solution to the Minimum Capacitary Vertex Cover problem is unique.

**Note.** It is sometimes useful to have  $\varphi(x)$  parametrized by the ratio  $t = \frac{\varphi(x)}{x}$ . Given any  $t \in (0, \infty)$ , as  $f(x, tx)$  simplifies to  $x((1+t) \log(1+t) - t \log t)$ ,  $1 = f(x, xt)$  implies  $x = \frac{1}{(1+t) \log(1+t) - t \log t}$ . It is easy to check that with decreasing  $t \in (0, \infty)$  this  $x \in (0, \infty)$  increases. So  $\partial T(1)$  parametrized by  $t = \frac{\varphi(x)}{x}$  reads as

$$\partial T(1) = \left\{ \left( \frac{1}{(1+t) \log(1+t) - t \log t}, \frac{t}{(1+t) \log(1+t) - t \log t} \right) : t \in (0, \infty) \right\}.$$

Moreover, differentiating  $(x + \varphi(x)) \log(x + \varphi(x)) - x \log x - \varphi(x) \log \varphi(x) = 1$  we can derive that

$$\varphi'(x) = -\frac{\log(1 + \frac{\varphi(x)}{x})}{\log(1 + \frac{x}{\varphi(x)})},$$

from which it is pretty obvious that  $\varphi' : (0, \infty) \rightarrow (-\infty, 0)$  smoothly increases from  $-\infty$  to 0 for  $x$  varying from 0 to  $+\infty$ .

Let  $G = (V, E)$  be a graph. It is clear that a minimum capacitary vertex cover  $x: V \rightarrow [0, \infty)$  for  $G$  achieves value 0 at each isolated vertex of  $G$ . Removing all isolated vertices of  $G$  (if any) will reduce the problem to the equivalent one on a graph without isolated vertices. The following lemma is a starting point to the study of exact solutions to the Minimum capacitary vertex cover problem.

**Lemma 2.** *Let  $G = (V, E)$  be a graph without isolated vertices and  $x: V \rightarrow (0, \infty)$  be a minimum capacitary vertex cover for  $G$ . Then the following hold:*

- (i)  $E^x := \{\{u, v\} \in E : f(x(u), x(v)) = 1\}$  is an edge cover of  $G$ .
- (ii) Let  $V_{<}^x := \{u \in V : x(u) < \frac{1}{2}\}$ ,  $V_{>}^x := \{u \in V : x(u) > \frac{1}{2}\}$ , and  $V_{=}^x := \{u \in V : x(u) = \frac{1}{2}\}$ . Then  $V_{<}^x$  is an independent set in  $G$  and  $N(V_{<}^x) = V_{>}^x$ .
- (iii) Every connected component  $F = (V(F), E(F))$  of the graph  $(V, E^x)$  is either the component of  $G[V_{=}^x]$ , or the component of the bipartite graph  $G[V_{<}^x, V_{>}^x]$ . In the latter case there exists  $q \in (0, \frac{1}{2})$  (depending on  $F$ ) such that

$$x(u) = \begin{cases} q, & \text{if } u \in V(F) \cap V_{<}^x \\ \varphi(q), & \text{if } u \in V(F) \cap V_{>}^x. \end{cases}$$

**Theorem 1.** *For any graph  $G = (V, E)$  the minimum capacitary vertex cover for  $G$  is unique.*

### Auxiliary Two-Valued Capacitary Vertex Cover

While for any graph  $G = (V, E)$  the function  $x \equiv \frac{1}{2}$  on  $V$  is always a feasible capacitary vertex cover, there are situations when this choice of  $x$  would clearly be suboptimal; for example, if  $G$  contains isolated vertices.

In view of part (iii) of Lemma 2, it is interesting to consider (minimum) capacitary vertex covers that assume exactly two different values on the vertex set of a graph without isolated vertices. More specifically, let  $q < \frac{1}{2}$  and  $p = \varphi(q) > \frac{1}{2}$  be these two values of a minimum capacitary vertex cover  $x$ , and let  $I \cup H$  be the partition of  $V$  such that  $x|_I = q$  and  $x|_H = p$ . Then, clearly,  $I$  is an independent set in  $G$  and we show that  $|I| > \frac{|V|}{2}$ .

More generally, for any partition  $I \cup H$  of  $V$  into two nonempty sets we can consider the following auxiliary two-valued minimization problem  $s := \min\{q|I| + p|H| : (q, p) \in T(1)\}$ . Let us consider the lines  $L_R = \{(q, p) : q|I| + p|H| = R\}$  for any real  $R$ . They all have a slope  $-\frac{|I|}{|H|}$  and exactly one of them touches the boundary of  $T(1)$ . The point of this touching,  $(x, \varphi(x))$  has to satisfy  $\varphi'(x) = -\frac{|I|}{|H|}$ .

As we observed earlier,  $\varphi'(x) = -\frac{\log(1+\frac{\varphi(x)}{x})}{\log(1+\frac{x}{\varphi(x)})}$ , so in terms of the parameter  $t = \frac{\varphi(x)}{x}$ , this point of touching corresponds to the unique root  $t$  (denoted as  $t = F(\frac{|H|}{|I|})$ , as it depends on the ratio  $\frac{|H|}{|I|}$  only) to the equation  $\frac{\log(1+t)}{\log(1+\frac{1}{t})} = \frac{|I|}{|H|}$  (or, equivalently,  $1 - \frac{|H|}{|I|} = \frac{\log t}{\log(t+1)}$ ).

The optimal solution  $s$  to the above minimization problem will then read in terms of  $F(\frac{|H|}{|I|})$  as

$$s = \frac{|I|}{\log(1 + F(\frac{|H|}{|I|}))} = \frac{|H|}{\log(1 + \frac{1}{F(\frac{|H|}{|I|})})}.$$

In another words, this minimum is the unique real root  $s$  to the equation  $2^{-\frac{|H|}{s}} + 2^{-\frac{|I|}{s}} = 1$ . In addition to  $F(\frac{|H|}{|I|})$ , we will denote by  $\Psi(\frac{|H|}{|I|})$  that  $(q, p)$  that minimizes  $s$  above, hence the point of touch. Namely,  $\Psi(\frac{|H|}{|I|})$  is the point

$$\left( \frac{\frac{1}{(1+F(\frac{|H|}{|I|}))\log(1+F(\frac{|H|}{|I|}))}-F(\frac{|H|}{|I|})\log F(\frac{|H|}{|I|})}{(1+F(\frac{|H|}{|I|}))\log(1+F(\frac{|H|}{|I|}))}, \frac{F(\frac{|H|}{|I|})}{(1+F(\frac{|H|}{|I|}))\log(1+F(\frac{|H|}{|I|}))}-F(\frac{|H|}{|I|})\log F(\frac{|H|}{|I|}) \right),$$

the unique point on  $\partial T(1)$  with the slope  $-\frac{|I|}{|H|}$ .

This auxiliary problem makes perfect sense for any partition  $I \cup H$  of  $V$  into two nonempty sets, but it is related to the minimum capacitary vertex cover problem only under some additional assumptions. By symmetry, we could confine ourselves to the case when  $|I| \geq |H|$ , so that point  $(q, p) = \Psi(\frac{|H|}{|I|})$  of  $\partial T(1)$  with the slope  $-\frac{|I|}{|H|}$  will have  $q \leq \frac{1}{2}$ . As  $|I| = |H|$  corresponds to the touching point  $(p, q) = (\frac{1}{2}, \frac{1}{2})$ , to achieve a two-valued solution we assume  $|I| > |H|$ , leading to  $q < \frac{1}{2}$ .

If one defines  $x|_I = q$  and  $x|_H = p$  then it will be a feasible capacitary vertex cover only if  $I$  is an independent set in  $G$ , otherwise the constraints  $(x(u), x(v)) \in T(1)$  for any edge with both vertices  $u$  and  $v$  in  $I$ , are not met.

We can then observe that a necessary condition for a graph  $G = (V, E)$  to have a two-valued minimum capacitary vertex cover is to have an independent set  $I$  of size  $> \frac{|V|}{2}$ . Later (in Theorem 4) we will be able to provide necessary and sufficient conditions for  $G$  to have a two valued minimum capacitary vertex cover.

## Minimum Fractional and Capacitary Vertex Covers

It is easy to see that any capacitary vertex cover for  $G = (V, E)$  is a fractional vertex cover for  $G$  (as  $T(1) \subseteq T$ , see Fig. 1), so  $vc_*(G) \leq vc_C(G)$ . Moreover, the uniform function  $x \equiv \frac{1}{2}$  on  $V$  is always a feasible capacitary vertex cover. Consequently,  $vc_*(G) \leq vc_C(G) \leq \frac{|V|}{2}$ . In terms of the conjunctive capacity,

$$\frac{2}{|V|} \leq C_{\text{AND}}(G) \leq \frac{1}{vc_*(G)}.$$

The Hallian graphs, i.e. those graphs that contain a system of vertex disjoint edges and (odd) cycles covering all the vertices are extremals for these bounds. Namely, assuming that a graph  $G = (V, E)$  does not have isolated vertices,  $vc_*(G) = \frac{|V|}{2}$  if and only if  $G$  is Hallian.

In the following theorem we prove that a similar result also holds for the Minimum Capacitary Vertex problem.

**Theorem 2.** Let  $G = (V, E)$  be a graph without isolated vertices, then  $\text{vc}_C(G) = \frac{|V|}{2}$  if and only if  $G$  is Hallian.

## 6 The Structure of Minimum Capacitary Vertex Covers

Now we describe in detail how the unique minimum capacitary vertex cover  $x : V \rightarrow (0, \infty)$  for a graph  $G = (V, E)$  without isolated vertices can look like. Let  $(I, H, K)$  be the canonical strong crown decomposition of  $G$ . We show that then  $V_<^x = I$ ,  $V_>^x = H$ , and  $V_=_^x = K$ . Moreover, we describe how the exact values of  $x$  on  $I \cup H$  can be found by computing binding sets of certain graphs.

**Theorem 3.** Let  $G = (V, E)$  be a graph without isolated vertices with the canonical strong crown decomposition  $(I, H, K)$  and let  $x : V \rightarrow (0, \infty)$  be the minimum capacitary vertex cover for  $G$ . Assume that  $x|_H$  achieves  $m \geq 1$  values,  $p_1 > p_2 > \dots > p_m$ . Then  $p_m > \frac{1}{2}$ , and  $x|_I$  achieves  $m$  values  $q_1 = \varphi(p_1) < q_2 = \varphi(p_2) < \dots < q_m = \varphi(p_m) < \frac{1}{2}$ .

Put  $I_i := \{v \in I : x(v) = q_i\}$ ,  $H_i := \{v \in H : x(v) = p_i\}$  for  $i = 1, 2, \dots, m$ . Then  $H_i \subseteq N(I_i) \subseteq \cup_{j=1}^i H_j$  and  $I_i \subseteq N(H_i) \cap I \subseteq \cup_{j=i}^m I_j$  for each  $i = 1, 2, \dots, m$ .

The ratio  $\frac{|H_i|}{|I_i|}$  is uniquely determined by the values  $(q_i, p_i)$ , namely  $(q_i, p_i) = \Psi(\frac{|H_i|}{|I_i|})$ . Moreover, for each connected component  $F = (V(F), E(F))$  of the graph  $(V, E^x)$  with  $V(F) \subseteq I_i \cup H_i$  we have the same ratio  $\frac{|V(F) \cap H_i|}{|V(F) \cap I_i|} = \frac{|H_i|}{|I_i|}$ , as  $(q_i, p_i) = \Psi(\frac{|V(F) \cap H_i|}{|V(F) \cap I_i|})$  holds.

In particular,  $\frac{|H_1|}{|I_1|} < \frac{|H_2|}{|I_2|} < \dots < \frac{|H_m|}{|I_m|} < 1$ .

We are now ready to describe graphs  $G = (V, E)$  for which a two-valued capacitary vertex cover is optimal. Such graphs have their canonical strong crown decomposition  $(I, H, K)$  with the Hallian part  $K$  empty and with  $I$  a binding set of  $G$ . In other words one could describe such graphs  $G = (V, E)$  as those with  $\alpha(G) > \frac{|V|}{2}$  and such that for each non-empty independent set  $U$  in  $G$   $\frac{|N(U)|}{|U|} \geq \frac{\tau(G)}{\alpha(G)}$ . Here  $\alpha(G)$  is the size of the maximum independent set and  $\tau(G)$  is the size of the minimum vertex cover in the graph  $G$ .

**Theorem 4.** Let  $G = (V, E)$  be a graph without isolated vertices and with the canonical strong crown decomposition  $(I, H, \emptyset)$  and assume that  $I \neq \emptyset$  is a binding set of  $G$ . Then the minimum capacitary vertex cover  $x : V \rightarrow (0, \infty)$  is two-valued,  $x|_I = q$ ,  $x|_H = p$ , with  $(q, p) = \Psi(\frac{|H|}{|I|})$ .

Let  $G = (V, E)$  be a graph without isolated vertices, let  $(I, H, K)$  be its canonical strong crown decomposition and assume that  $I \neq \emptyset$  (so  $\text{bind}(G) < 1$ ). Let  $I_1$  be the inclusionwise maximal binding set of  $G$ . As we observed earlier  $I_1 \subseteq I$ . Put  $H_1 := N(I_1)$  and  $(q_1, p_1) = \Psi(\frac{|H_1|}{|I_1|})$ . Let us take  $x|_{I_1} = q_1$ ,  $x|_{H_1} = p_1$ . In the induced graph  $G[I_1 \cup H_1]$  we can conclude by the previous theorem that the

minimum capacitary vertex cover is two-valued and achieves the values we have assigned them.

Now we study the question on the graph obtained by removing  $I_1$  and  $H_1$  from  $G$ . Denote  $G_1 = G[V \setminus (I_1 \cup H_1)]$ . It can be easily verified that  $G_1$  has the canonical strong crown decomposition  $(I \setminus I_1, H \setminus H_1, K)$ . (It is sufficient to check that it is still  $|N(U) \cap I| > |U|$  whenever  $U$  is a nonempty subset of  $H \setminus H_1$ .)

If  $I \setminus I_1 \neq \emptyset$  (so  $\text{bind}(G_1) < 1$ ) we can take  $I_2$ , the inclusionwise maximal binding set of  $G_1$ ,  $H_2 := N_{G_1}(I_2)$  and put  $(q_2, p_2) = \Psi(\frac{|H_2|}{|I_2|})$ . We take  $x|_{I_2} = q_2$ ,  $x|_{H_2} = p_2$ . With this choice of  $x$  in the induced graph  $G[I_2 \cup H_2]$ ,  $x$  defines the minimum capacitary vertex cover.

Observe now that  $\text{bind}(G) = \frac{|H_1|}{|I_1|} < \frac{|H_2|}{|I_2|} = \text{bind}(G_1)$ . If not, then  $\frac{|H_1 \cup H_2|}{|I_1 \cup I_2|} \leq \text{bind}(G)$  and  $I_1 \cup I_2$  would be an inclusionwise larger binding set of  $G$  than  $I_1$ , a contradiction. So

$$\frac{|H_1|}{|I_1|} < \frac{|H_2|}{|I_2|}, \text{ and as } (q_i, p_i) = \Psi\left(\frac{|H_i|}{|I_i|}\right), i = 1, 2, \dots$$

we conclude that  $q_1 < q_2 < \frac{1}{2}$  and  $p_1 > p_2 > \frac{1}{2}$ .

We can continue in the same way and stop after  $m$  steps once  $\cup_{i=1}^m I_i$  exhausts  $I$ . The rest is the Hallian graph  $G[K]$ . For  $K$  we take  $x|_K \equiv \frac{1}{2}$  as we know that for Hallian graphs it is the optimal value of the minimum capacitary vertex cover (Theorem 2). We have

$$\frac{|H_1|}{|I_1|} < \frac{|H_2|}{|I_2|} < \dots < \frac{|H_m|}{|I_m|} < 1, (q_i, p_i) = \Psi\left(\frac{|H_i|}{|I_i|}\right), x|_{I_i} = q_i, x|_{H_i} = p_i,$$

for  $i = 1, 2, \dots, m$  and  $q_1 < q_2 < \dots < q_m < \frac{1}{2}$ .

If we define  $x: V \rightarrow (0, \infty)$  this way, we have a potential solution that is locally optimal on each of  $G[I_1 \cup H_1]$ ,  $G[I_2 \cup H_2]$ ,  $\dots$ ,  $G[I_m \cup H_m]$ , and  $G[K]$ . So  $\text{vc}_C(G)$  is at least  $x(V)$  and to show that it is indeed  $x(V)$  we need to check that  $x$  is a feasible capacitary vertex cover for  $G$  satisfying all the constraints  $(x(u), x(v)) \in T(1)$  also for pairs  $\{u, v\} \in E$  belonging to distinct pieces of  $G[I_1 \cup H_1]$ ,  $G[I_2 \cup H_2]$ ,  $\dots$ ,  $G[I_m \cup H_m]$ , and  $G[K]$ . For this it is sufficient to check that all vertices of  $I$  that have assigned  $x(u) < \frac{1}{2}$ , have their neighbours  $v$  with  $x(v)$  large enough. If  $u \in I_i$ , say, then  $x(u) = q_i$ , and any neighbour  $v$  of  $u$  needs to have  $x(v) \geq \varphi(q_i) = p_i$  to comply with the constraints, so only  $v \in \cup_{j=1}^i H_j$  would be allowed. But our construction ensures that  $H_i \subseteq N(I_i) \subseteq \cup_{j=1}^i H_j$  for each  $i = 1, 2, \dots, m$ , so there are no edges from  $u \in I_i$  to any  $v \in H_j$  with  $j > i$ . Hence our  $x$  is indeed a feasible capacitary vertex cover for  $G$  and consequently it is the minimum capacitary vertex cover for  $G$ .

**Theorem 5.** *Let  $G = (V, E)$  be a graph without isolated vertices and with the canonical strong crown decomposition  $(I, H, K)$ , and let  $x: V \rightarrow (0, \infty)$  be the (unique) minimum capacitary vertex cover for  $G$ . Then  $I = \{v \in V : x(v) < \frac{1}{2}\}$ ,  $H = \{v \in V : x(v) > \frac{1}{2}\}$  and  $K = \{v \in V : x(v) = \frac{1}{2}\}$ . If  $G$  is not Hallian, then  $x|_I$  has  $m \geq 1$  values  $q_1 < q_2 < \dots < q_m < \frac{1}{2}$  and  $x|_H$  has  $m$  values  $p_1 = \varphi(q_1) > p_2 = \varphi(q_2) > \dots > p_m = \varphi(q_m) > \frac{1}{2}$  with  $I_i := \{u \in I : x(v) = q_i\}$ ,*

$H_i := \{v \in H : x(v) = p_i\}$ . Here  $I_1$  is the inclusionwise maximal binding set for  $G$ ,  $H_1 = N(I_1)$  and  $(q_1, p_1) = \Psi(\frac{|H_1|}{|I_1|})$ . Moreover, after removing the vertices  $I_1 \cup H_1$  from the graph,  $x|_{V \setminus (I_1 \cup H_1)}$  is still a minimum capacitary vertex cover in the rest  $G[V \setminus (I_1 \cup H_1)]$ . We can find all pairs  $(I_i, H_i)$  of sets and their corresponding values  $(q_i, p_i)$ ,  $i = 1, 2, \dots, m$ , by repeatedly computing the inclusionwise maximal binding set  $I_i$  in  $G_{i-1} := G[V \setminus \cup_{j < i} (I_j \cup H_j)]$ ,  $H_i = N_{G_{i-1}}(I_i)$ , and  $(q_i, p_i) = \Psi(\frac{|H_i|}{|I_i|})$ .

*Remark 2.* As the convex programming problem of finding the minimum capacitary vertex cover problem is efficiently computable one can use it as an alternative way of computing the canonical strong crown decomposition  $(I, H, K)$  of any graph  $G = (V, E)$  without isolated vertices, given that  $I = V^x_<$ ,  $H = V^x_>$ , and  $K = V^x_=$ .

## References

1. Berge, C.: Graphs. North-Holland, Amsterdam (1985)
2. Chlebík, M., Chlebíková, J.: Crown reductions for the Minimum Weighted Vertex Cover problem. Discrete Applied Mathematics 156, 292–312 (2008)
3. Cohen, G., Körner, J., Simonyi, G.: Zero-error capacities and very different sequences. In: Capocelli, R.M. (ed.) Sequences: Combinatorics, Compression, Security and Transmission, pp. 144–155. Springer (1990)
4. Cunningham, W.H.: Computing the binding number of a graph. Discr. Appl. Math 27, 283–285 (1990)
5. Gargano, L., Körner, J., Vaccaro, U.: Sperner capacities. Graphs and Combinatorics 9, 31–46 (1993)
6. Gargano, L., Körner, J., Vaccaro, U.: Capacities: from information theory to extremal set theory. J. Comb. Theory Ser. A 68, 296–316 (1994)
7. Greco, G.: Capacities of graphs and 2-matchings. Discrete Mathematics 186, 135–143 (1998)
8. Kane, V.G., Mohanty, S.P., Straus, E.G.: Which rational numbers are binding numbers? Journal of Graph Theory 5, 379–384 (1981)
9. Körner, J., Simonyi, G.: A Sperner-type theorem and qualitative independence. J. Comb. Theory 59, 90–103 (1992)
10. Rényi, A.: Probability theory. North-Holland, Amsterdam (1970)
11. Shannon, C.E.: The zero-error capacity of a noisy channel. IRE Trans. Inform. Theory 2, 8–19 (1956)
12. Woodal, D.R.: The binding number of a graph and its Anderson number. J. Combin. Theory Ser. B 15, 225–255 (1973)

# Improved Approximation Algorithms for the Facility Location Problems with Linear/submodular Penalty

Yu Li<sup>1</sup>, Donglei Du<sup>2</sup>, Naihua Xiu<sup>1</sup>, and Dachuan Xu<sup>3,\*</sup>

<sup>1</sup> Department of Mathematics, School of Science, Beijing Jiaotong University, 3  
Shangyuancun, Haidian District, Beijing 100044, P.R. China

<sup>2</sup> Faculty of Business Administration, University of New Brunswick, NB Canada  
Fredericton E3B 9Y2

<sup>3</sup> Department of Applied Mathematics, Beijing University of Technology, 100  
Pingleyuan, Chaoyang District, Beijing 100124, P.R. China  
[xudc@bjut.edu.cn](mailto:xudc@bjut.edu.cn)

**Abstract.** We consider the *facility location problem with submodular penalty* (FLPSP) and the *facility location problem with linear penalty* (FLPLP), two extensions of the classical *facility location problem* (FLP). First, we introduce a general algorithmic framework for a class of covering problems with *submodular* penalty, extending the recent result of Geunes et al. [11] with *linear* penalty. This framework leverages existing approximation results for the original covering problems to obtain corresponding results for their submodular penalty counterparts. Specifically, any LP-based  $\alpha$ -approximation for the original covering problem can be leveraged to obtain an  $1/\left(1-e^{-1/\alpha}\right)$ -approximation algorithm for the counterpart with submodular penalty. Consequently, any LP-based approximation algorithm for the classical FLP (as a covering problem) can yield, via this framework, an approximation algorithm for the submodular penalty counterpart. Second, by exploiting some special properties of FLP, we present an LP rounding algorithm which has the currently best 2-approximation ratio over the previously best 2.50 by Hayrapetyan et al. [13] for the FLPSP and another LP rounding algorithm which has the currently best 1.5148-approximation ratio over the previously best 1.853 by Xu and Xu [27] for the FLPLP, respectively.

**Keywords:** Approximation algorithm, facility location problem, LP rounding, submodular function.

## 1 Introduction

The classical *facility location problem* (FLP) is one of the most important models in combinatorial optimization with applications in operations research and computer science in general, and inventory management and supply chain management in particular. This problem is NP-hard and therefore much attention

---

\* Corresponding author.

has been focused on designing approximation algorithms with good performance. Following the first constant-factor approximation algorithm by Shmoys et al. [22], there was a long list of work on designing improved approximation algorithms for this problem over the years. As a result, four basic schemes have emerged in the design of these approximation algorithms, namely *LP-rounding* [3,8,18,22,24], *primal-dual* [16], *dual-fitting* [14,15,20], and *local search* [4,12,17]. These four competitive and complementary schemes possess different features. LP-rounding is *non-combinatorial* in nature mainly because of the resolving of the underlying LP-relaxation, while the other three are *combinatorial*, hence offering faster approximation algorithms than LP rounding. However, the first scheme usually allows us to design algorithms with better approximation ratios compared to the other three—e.g., the currently best approximation ratio of 1.488 by Li [18] is obtained by combining LP-rounding and dual-fitting algorithms. Among the latter three, primal-dual can be adapted to solve variants of the classical FLP. Dual-fitting is essentially one special type of primal-dual methods, and usually offers better approximation ratio than a typical primal-dual, but is less robust. Local search is more powerful on the hard capacitated version of the FLP (cf. [30]).

On the impossibility of approximation, Guha and Khuller [12] show that the approximation ratio for FLP is at least 1.463, unless  $NP \subseteq DTIME(n^{\log \log n})$ , later strengthened to unless  $P = NP$  by Sviridenko ([25]). For other variants of the FLP, we refer to [1,2,6,19,21,23,28,29,30] and the references therein.

In this work, we study two facility location problems with penalty, namely the *facility location problem with linear penalty* (FLPLP) and the more general *facility location problem with submodular penalty* (FLPSP), first introduced, as important extensions of the classical FLP, by Charikar et al. [5] and Hayrapetyan et al. [13], respectively. The FLPLP can be formally defined as follows. Consider a set  $\mathcal{F}$  of facilities and a set  $\mathcal{D}$  of clients. For every facility  $i \in \mathcal{F}$  and every client  $j \in \mathcal{D}$ , there is a nonnegative opening cost  $f_i$ , a penalty cost  $p_j$ , and a connection cost  $c_{ij}$ . Unlike the regular FLP, the FLPLP does not require that every client must be served by some open facility. Instead, a client  $j$  can be either served by an open facility or rejected for service with penalty cost  $p_j$ . The problem is to open a subset of facilities such that each client  $j \in \mathcal{D}$  is either assigned to an open facility or rejected with the objective to minimize the total cost, including the open, connection and linear penalty costs. We assume that the connection cost between clients and facilities is metric, i.e.,  $c_{ij} \leq c_{ij'} + c_{i'j'} + c_{i'j}$ . The FLPSP is similar to the FLPLP except that the linear penalty cost is replaced by a general monotonically increasing submodular function. A set function  $P : 2^{\mathcal{D}} \rightarrow \mathbb{R}_+$  is *submodular* if  $P(X \cap Y) + P(X \cup Y) \leq P(X) + P(Y)$  for any subsets  $X, Y \in 2^{\mathcal{D}}$ . We assume that  $P(\emptyset) = 0$ .

For the FLPLP, four constant-factor approximation algorithms were known in the literature. Charikar et al. [5] gave a primal-dual 3-approximation algorithm. Xu and Xu [26,27] presented an LP-rounding based  $2 + e^{-1} \approx 2.736$ -approximation algorithm, and a combinatorial 1.853-approximation algorithm by integrating primal-dual with local search technique. Recently, Geunes et al.

[11] gave an improved LP-rounding based 2.056-approximation algorithm. In particular, Geunes et al. [11] presented an algorithmic framework which can convert any LP-based  $\alpha$ -approximation for the classical FLP to an  $(1 - e^{-1/\alpha})^{-1}$ -approximation algorithm for the counterpart with linear penalty.

For the FLPSP, three approximation algorithms were proposed in the literature. Hayrapetyan et al. [13] gave a simple LP-rounding based 2.50-approximation algorithm. Chudak and Nagano [7] gave a faster  $(2.50 + \epsilon)$ -approximation algorithm by solving a convex relaxation rather than an LP relaxation of the FLPSP. Very recently, Du et al. [9] presented a primal-dual 3-approximation algorithm.

In summary, for the FLPLP, the best known approximation ratio is 1.853 [27], and the best known non-combinatorial ratio is 2.056 [11]. For the FLPSP, the best known combinatorial approximation ratio is 3 [9] and the best known non-combinatorial ratio is 2.50 [13].

The main contributions of this work are summarized as follows.

- We extend Geunes et al. [11]’s algorithmic framework for linear penalty to submodular penalty by showing that our framework can leverage any LP-based  $\alpha$ -approximation to construct an  $(1 - e^{-1/\alpha})^{-1}$ -approximation algorithm for the counterpart with submodular penalty.
- Combining a novel LP-rounding technique with the JMS algorithm of [14,15], along with the exploitation of the special properties of the submodular penalty function, we provide an improved 2-approximation algorithm for the FLPSP over the previously best approximation ratio 2.50 [13].
- Note that for the FLPLP, the existing combinatorial ratio is better than the existing non-combinatorial ratio. This phenomena is generally “abnormal” and our third contribution corrects this “abnormality” by offering the currently best LP-rounding 1.5148-approximation algorithm which exploits the special properties of the linear penalty function.

There are intrinsic differences between the two LP-based algorithms for the FLPSP in Section 2 and the FLPLP in Section 3 due to the essential difference between linear and submodular functions. In general, linear penalty functions possess important properties not applicable to submodular penalty functions. Our algorithm and analysis indicate that the latter is substantially harder to approximate than the former because the techniques for the FLPLP, such as that by Byrka and Aardal [3], is not directly applicable to the FLPSP. To overcome this difficulty, our algorithm for the latter will exploit the special structure of the optimal LP relaxation solution of the FLPSP.

The rest of this paper is organized as follows. In Section 2, we first show that any LP-based  $\alpha$ -approximation algorithm for the classical FLP can be leveraged to an  $(1 - e^{-1/\alpha})^{-1}$ -approximation algorithm for the submodular penalty counterpart. This result will serve as a concrete example in deriving a general framework for a class of covering problems with submodular penalty. And this framework also extends Geunes et al. [11]’s technique for the linear penalty model to the submodular case. In Sections 3 and 4, we present the 2-approximation

algorithm for the FLPSP, and the 1.5148-approximation algorithm for the FLLP, respectively.

We use the following notations throughout the paper:  $n_f = |\mathcal{F}|$  and  $n_c = |\mathcal{D}|$ . In the rest of this paper, all proofs are omitted and deferred to the journal version.

## 2 Algorithmic Scheme for Problems with Submodular Penalty

We will use the FLPSP as an example to show our algorithmic framework, which then will be extended to a more general class of covering problems.

### 2.1 An LP-Rounding Approximation Algorithm for the FLPSP

The following LP relaxation for the FLPSP first appeared in Hayrapetyan et al. [13].

$$\begin{aligned} \min \quad & \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{D}} c_{ij} x_{ij} + \sum_{i \in \mathcal{F}} f_i y_i + \sum_{S \subseteq \mathcal{D}} P(S) z_S \\ \text{s.t.} \quad & \sum_{i \in \mathcal{F}} x_{ij} + \sum_{S \subseteq \mathcal{D}: j \in S} z_S \geq 1, \quad \forall j \in \mathcal{D}, \\ & x_{ij} \leq y_i, \quad \forall i \in \mathcal{F}, j \in \mathcal{D}, \\ & x_{ij}, y_i, z_S \geq 0, \quad \forall i \in \mathcal{F}, j \in \mathcal{D}, S \subseteq \mathcal{D}, \end{aligned} \quad (1)$$

where  $P(S)$  is a nondecreasing submodular function and  $P(\emptyset) = 0$ . Let us give some intuitive interpretation for the corresponding integer program as follows. The binary variables  $x_{ij}$  indicates whether client  $j$  is served by facility  $i$ ;  $y_i$  indicates whether facility  $i$  is open;  $z_S$  indicates whether client set  $S$  is penalized, respectively. The first set of constraints says that each client  $j$  is served by some facility or rejected at some client set. The second set of constraints says that each client  $j$  can only be served by an open facility.

Now, we give an LP-rounding based approximation algorithm similar to that by Geunes et al. [11].

#### Algorithm 1.

**STEP 1.** Solve the LP relaxation (1) to obtain an optimal fractional solution  $(x^*, y^*, z^*)$ .

**STEP 2.** Construct a new variable  $z$  such that  $z_j := 1 - \sum_{i \in \mathcal{F}} x_{ij}^*$ .

**STEP 3.** Select parameter  $\beta$  uniformly at random from the interval  $[0, \delta]$ .

**STEP 4.** Reject the subset  $S := \{j | z_j \geq \beta\}$ , and pay the penalty cost  $P(S)$ .

**STEP 5.** Construct an instance of the classical FLP with the set of facilities

$\mathcal{F}$ , the set of clients  $\mathcal{D} \setminus S$  and the connection cost  $c_{ij}$  ( $i \in \mathcal{F}$ , and  $j \in \mathcal{D} \setminus S$ ).

Then run the 1.488-approximation algorithm [18] for the instance and assign the clients in  $\mathcal{D} \setminus S$  to the closest open facilities.

**Lemma 1.** *The expected penalty cost of the solution generated by Algorithm 1 is no more than  $\delta^{-1} \sum_{S \subseteq \mathcal{D}} P(S) z_S^*$ .*

**Theorem 2.** *Setting  $\delta = 1 - e^{-1/1.488}$ , the approximation ratio of Algorithm 1 for the FLPSP is no more than  $(1 - e^{-1/1.488})^{-1} \leq 2.044$ .*

We remark that there is a convex relaxation for the FLPSP presented by Chudak and Nagano [7],

$$\begin{aligned} \min & \sum_{i \in \mathcal{F}} \sum_{j \subseteq \mathcal{D}} c_{ij} x_{ij} + \sum_{i \in \mathcal{F}} f_i y_i + P'(z) \\ \text{s.t. } & \sum_{i \in \mathcal{F}} x_{ij} + z_j \geq 1, \quad \forall j \in \mathcal{D}, \\ & x_{ij} \leq y_i, \quad \forall i \in \mathcal{F}, j \in \mathcal{D}, \\ & x_{ij}, y_i, z_j \geq 0, \quad \forall i \in \mathcal{F}, j \in \mathcal{D}, \end{aligned} \quad (3)$$

in which,

$$\begin{aligned} P'(z) = \max & \sum_{j \in \mathcal{D}} \alpha_j z_j \\ \text{s.t. } & \sum_{j \in S} \alpha_j \leq P(S), \quad \forall S \subseteq \mathcal{D}, \\ & \alpha_j \geq 0, \quad \forall j \in \mathcal{D}. \end{aligned}$$

It is not hard to see that in Algorithm 1 the solution  $(x^*, y^*, z)$  constructed in Step 2 is actually an optimal solution to (3). To avoid solving (1) with exponential number of variables, we can replace the first two steps in Algorithm 1 by just solving the convex relaxation (3) to obtain a new algorithm with the same approximation ratio except the extra term  $\varepsilon$ .

## 2.2 General Rounding Framework

We can extend our rounding technique in the previous section to the following more general model:

$$\begin{aligned} \min & \varphi(w) + \sum_{S \subseteq \mathcal{D}} P(S) z_S \\ \text{s.t. } & w_j + \sum_{S \subseteq \mathcal{D}: j \in S} z_S \geq 1, \quad \forall j \in \mathcal{D}, \\ & w_j, z_S \in \{0, 1\}, \quad \forall j \in \mathcal{D}, S \subseteq \mathcal{D}. \end{aligned} \quad (4)$$

This model captures a class of covering problems, in which the clients can receive service or be rejected, and the subset of clients  $S_{rej}$  rejected will incur a penalty cost  $P(S_{rej})$ . The binary vector  $w$  indicates whether the client  $j$  receives service or not. The term  $\varphi$  in the objective function satisfies Assumption 3 below. This new problem embeds a subproblem, denoted as  $\phi(w)$ .

**Assumption 3.** There exists a function  $\bar{\varphi}: [0, 1]^{n_c} \mapsto R_+$ , such that

- 1)  $\bar{\varphi}$  is a lower bound on  $\varphi$ , i.e.  $\bar{\varphi}(w) \leq \varphi(w)$ , for all  $w \in [0, 1]^{n_c}$ ;
- 2) for any fixed  $w \in \{0, 1\}^{n_c}$ , we can efficiently find a solution to  $\phi(w)$  of cost at most  $\alpha\bar{\varphi}(w)$ , where  $\alpha \geq 1$ ;
- 3) the optimization problem

$$\begin{aligned} \min \quad & \bar{\varphi}(w) + \sum_{S \subseteq \mathcal{D}} P(S) z_S \\ \text{s.t.} \quad & w_j + \sum_{S \subseteq \mathcal{D}: j \in S} z_S = 1, \quad \forall j \in \mathcal{D}, \\ & w_j, z_S \in [0, 1], \quad \forall j \in \mathcal{D}, S \subseteq \mathcal{D}, \end{aligned} \tag{5}$$

can be solved efficiently.

In general the subproblem  $\varphi(w)$  is NP-hard—e.g., for the FLPSP,  $\phi(w)$  is the classical FLP. The above assumption will make it possible for us to design a constant-factor approximation algorithm for the general model. Note that (5) is a relaxation of (4), and becomes (1) in the case of the FLPSP, for example.

The following algorithm is an extension of Algorithm 1.

### Algorithm 2.

**STEP 1.** Solve the relaxation problem (5) to obtain an optimal fractional solution  $(w^*, z^*)$ .

**STEP 2.** Select the parameter  $\beta$  uniformly at random from the interval  $[0, \delta]$ .

**STEP 3.** Reject the subset  $S := \{j | 1 - w_j^* \geq \beta\}$ , and pay the penalty cost  $P(S)$ . Construct variable  $w \in \{0, 1\}^{n_c}$  by setting  $w := I(\mathcal{D} \setminus S)$ .

**STEP 4.** Find a solution to the subproblem  $\phi(w)$  and serve all the unrejected clients  $\mathcal{D} \setminus S$ .

We need one more assumption called *scaling* property [11].

**Assumption 4.** The function  $\bar{\varphi}$  satisfies the scaling property if

$$\bar{\varphi}(w) \leq \frac{1}{1 - \beta} \bar{\varphi}(w^*), \quad \forall w^* \in [0, 1]^{n_c}, \forall 0 \leq \beta < 1.$$

Note that this assumption indeed holds in the FLPSP, because  $(x^*/(1 - \beta), y^*/(1 - \beta))$  is a feasible solution to the relaxed problem.

With Assumptions 3 and 4, similar proofs to those in Lemma 1 and Theorem 2 lead to the following result:

**Theorem 5.** Setting  $\delta = 1 - e^{-1/\alpha}$ , the approximation ratio of Algorithm 2 for (4) is no more than  $(1 - e^{-1/\alpha})^{-1}$ .

### 3 Improved 2-Approximation Algorithm for the FLPSP

In the previous section, we presented a general algorithmic frame for a class of covering problems with submodular penalty, but this frame can be too crude to be applied to certain specific problems and hence sometimes we need to refine the general frame to yield improved approximation ratios. For example, when we applied it to the FLPSP earlier in Section 2.1, the algorithm and its analysis ignored the possibility where unrejected clients may have paid fractional cost for penalty. In this section, we will incorporate this ignored cost into the design and analysis of our algorithms to obtain improved approximation ratio for the FLPSP.

#### Algorithm 3.

**STEP 1.** Solve the convex relaxation (2) of the problem to obtain an optimal fractional solution  $(x^*, y^*, z^*)$ . Or solve the LP relaxation (1) and convert its solution into  $(x^*, y^*, z^*)$  for (2).

**STEP 2.** Reject the subset  $S_r := \{j | z_j \geq \frac{1}{2}\}$ , and pay the penalty cost  $P(S)$ .

**STEP 3.** Construct an instance of classical FLP with the set of facilities  $\mathcal{F}$ , the set of clients  $\mathcal{D} \setminus S_r$  and the connection cost  $c_{ij}$ ,  $i \in \mathcal{F}$ ,  $j \in \mathcal{D} \setminus S_r$ . Then run the JMS algorithm [14,15] for the instance and assign the clients in  $\mathcal{D} \setminus S_r$  to the closest open facilities.

We will show that Algorithm 3 offers an improved approximation ratio. After the Step 1 of Algorithm 3, we know the total cost of the optimal fractional solution  $(x^*, y^*, z^*)$  for (1) is

$$\sum_{i \in \mathcal{F}} \sum_{j \subseteq \mathcal{D}} c_{ij} x_{ij}^* + \sum_{i \in \mathcal{F}} f_i y_i^* + P'(z^*).$$

Moreover, we have

#### Lemma 6.

- (1)  $\sum_{j \in \mathcal{D}} \alpha_j^* z_j^* = P'(z^*)$ ;
- (2)  $P(S_r) \leq 2 \sum_{j \in S_r} \alpha_j^* z_j^*$ ;
- (3) For any  $j \notin S_r$ , if  $z_j^* > 0$  and  $x_{ij}^* > 0$ , then  $\alpha_j^* \geq c_{ij}$ .

We now present the approximation ratio of the above algorithm.

**Theorem 7.** *Algorithm 3 is a 2-approximation algorithm for the FLPSP.*

### 4 Improved 1.5148-Approximation Algorithm for the FLPLP

Recently, Li [18] offered an improved analysis of the LP rounding algorithm by Byrka and Aardal [3] to obtain the currently best approximation ratio for the

classical to FLP. In this section, we extend the LP rounding algorithm by Byrka and Aardal and the analysis by Li to obtain an improved algorithm for the FLPLP, which has a natural integer programming formulation:

$$\begin{aligned} \min \quad & \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{D}} c_{ij} x_{ij} + \sum_{i \in \mathcal{F}} f_i y_i + \sum_{j \in \mathcal{D}} p_j z_j \\ \text{s.t.} \quad & \sum_{i \in \mathcal{F}} x_{ij} + z_j \geq 1, \quad \forall j \in \mathcal{D}, \\ & x_{ij} \leq y_i, \quad \forall i \in \mathcal{F}, j \in \mathcal{D}, \\ & x_{ij}, y_i, z_j \in \{0, 1\}, \quad \forall i \in \mathcal{F}, j \in \mathcal{D}, \end{aligned} \tag{6}$$

where the binary variable  $x_{ij}$  indicates whether client  $j$  is connected to facility  $i$  or not, binary variable  $y_i$  indicates whether facility  $i$  is open or not, and binary variable  $z_j$  indicates whether client  $j$  is penalized or not.

Denote the optimal solution of the LP relaxation as  $(x^*, y^*, z^*)$  and the corresponding optimal dual solution as  $(\alpha^*, \beta^*)$ . The total optimal fractional cost includes three parts: the opening cost  $F^* = \sum_{i \in \mathcal{F}} f_i y_i^*$ , the connection cost  $C^* = \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{D}} c_{ij} x_{ij}^* = \sum_{j \in \mathcal{D}} C_j^*$ , and the penalty cost  $P^* = \sum_{j \in \mathcal{D}} p_j z_j^* = \sum_{j \in \mathcal{D}} P_j^*$ .

We have the following lemma to capture the property of the optimal fractional solution  $(x^*, y^*, z^*)$ .

**Lemma 8.** *For any  $j \in \mathcal{D}$ , if  $z_j^* > 0$  and  $x_{ij}^* > 0$ , then  $p_j \geq c_{ij}$ .*

For any given parameter  $\gamma > 1$ , we partition the set of clients  $\mathcal{D}$  into two subsets.

$$\begin{aligned} \mathcal{D}_\gamma &= \left\{ j \in \mathcal{D} \mid \sum_{i \in \mathcal{F}} x_{ij}^* \geq \frac{1}{\gamma} \right\}, \\ \bar{\mathcal{D}}_\gamma &= \mathcal{D} \setminus \mathcal{D}_\gamma. \end{aligned}$$

In our algorithm, we first randomly scale  $y^*$  by  $\gamma > 1$  to obtain a suboptimal fractional solution  $(x^*, \gamma y^*, z^*)$ , making room for modification of  $x^*$  and  $z^*$  in order to reduce the fractional connection cost and the fractional penalty cost respectively. Then, for each client  $j$ , we modify the corresponding  $x^*$  to connect facility as close as possible subject to that the summation  $\sum_{i \in \mathcal{F}} x_{ij}$  equals to  $\min\{\gamma \sum_{i \in \mathcal{F}} x_{ij}^*, 1\}$ , followed by modifying the corresponding  $z^*$  to guarantee the first constraint of (6), resulting in a new feasible solution. However, Lemma 8 implies that for any client  $j \in \mathcal{D}_\gamma$ , the new  $z$ -variable must be zero. So for  $j \in \mathcal{D}_\gamma$ , we can omit  $z$ -variables and let  $(\bar{x}, \bar{y})$  be the resultant *complete* solution (i.e. there exists no  $i \in \mathcal{F}$  and  $j \in \mathcal{D}$  such that  $0 < \bar{x}_{ij} < \bar{y}_i$ ; otherwise we may split the facilities to obtain an equivalent instance with a complete solution—refer to Lemma 1 in [24] for a more detailed argument). For a client  $j$ , we say that a facility  $i$  is one of its *close* facilities if it fractionally serves client  $j$  in  $(\bar{x}, \bar{y})$ . If  $\bar{x}_{ij} = 0$ , but facility  $i$  was serving client  $j$  in solution  $(x^*, y^*, z^*)$ , then we say that  $i$  is a *distant* facility of client  $j$ .

For any client  $j \in \mathcal{D}$ , let  $i_1, i_2, \dots, i_m$  be the facilities, arranged in the non-decreasing order of distances, to which client  $j$  is connected to fractionally in  $(x^*, y^*, z^*)$ . Then we define  $h_j(p) = c_{i_t, j}$ , where  $t$  is minimum number such that  $\sum_{s=1}^t y_{i_s}^* \geq p$ , or  $h_j(p) = p_j$  when  $p > \sum_{s=1}^m y_{i_s}^*$ .

Note that for every client  $j \in \mathcal{D}$ , the following facts hold:

- The sum of average connection and penalty costs equals  $D_{av} = \int_0^1 h_j(p) dp = F_j^* + P_j^*$ .
- The average connection cost to a close facility equals  $D_{av}^C = \gamma \int_0^{1/\gamma} h_j(p) dp$ .
- The average connection cost to a distant or rejected facility equals  $D_{av}^D = \frac{\gamma}{\gamma-1} \int_{1/\gamma}^1 h_j(p) dp$ .
- The maximal distance to a close facility is at most the average distance to a distant facility,  $D_{max}^C = h_j(1/\gamma)$ .

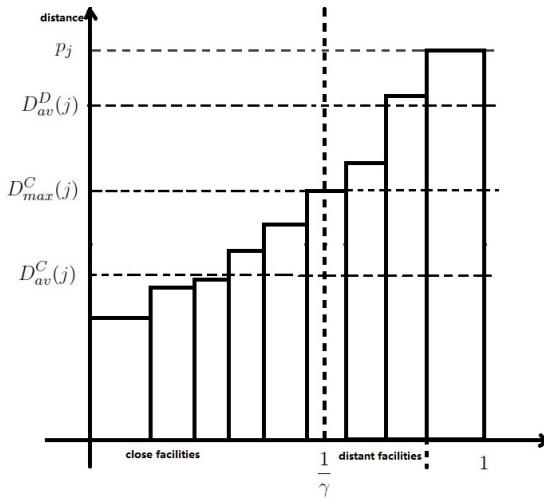


Fig. 1.

We are now ready to present our algorithm. Consider the bipartite graph  $G$  obtained from the solution  $(\bar{x}, \bar{y})$ , where each client  $j \in \mathcal{D}_\gamma$  is directly connected to his close facilities. Clients connected to the same facility in  $G$  are called *neighbors*. The main component in the algorithm is to cluster this graph recursively in a greedy fashion (Step 4), similarly to that used by Byrka and Aardal [3].

We let  $\gamma = 1.3360$  with probability 0.45 and with the remaining 0.55 probability, we choose  $\gamma$  between  $(1.3360, 1.9860]$  uniformly. We denote the distribution as  $\mu(\gamma)$ . The main idea on how to choose this distribution  $\mu(\gamma)$  is to first conduct numerical test to hypothesize the distribution function form such that the approximation ratio is as small as possible; and then the hypothesized functional form can be proved rigorously afterwards.

**Algorithm 4.**

- STEP 1.** Solve the LP relaxation (6) to obtain an optimal fractional solution  $(x^*, y^*, z^*)$ .
- STEP 2.** Scale up the value of the facility opening variables  $y^*$  by a random number  $\gamma$  obeying  $\mu(\gamma)$ . Then modify the value of the  $x^*, z^*$ -variables so that each client is connected to its closest fractionally open facilities.
- STEP 3.** If necessary, split facilities to obtain a complete solution  $(\bar{x}, \bar{y})$ .
- STEP 4.** Construct a greedy clustering from solution  $(\bar{x}, \bar{y})$  by choosing recursively as cluster centers the unclustered clients in  $\mathcal{D}_\gamma$  with minimal  $D_{av}^C(j) + D_{\max}^C(j)$ .
- STEP 5.** For every cluster center  $j$ , open one of its close facilities randomly with probabilities  $\bar{x}_{ij}$ .
- STEP 6.** For each facility  $i$  that is not a close facility of any cluster center, open it independently with probability  $\bar{y}_i$ .
- STEP 7.** For any client  $j \in \mathcal{D}$ , connect it to an open facility or pay penalty cost when the minimal connection cost to an open facility is larger than  $p_j$ .

We remark that only clients in  $\mathcal{D}_\gamma$  can be chosen as cluster centers, which is different from the LP rounding algorithm in Byrka and Aardal [3] and Li [18].

During the analysis of the above algorithm, we will use the following lemma from [18] which holds also for our algorithm.

**Lemma 9.** *For any client  $j$ , we have*

$$E[C_j + P_j] \leq \int_0^1 h_j(p) e^{-\gamma p} \gamma dp + e^{-\gamma} \left( \gamma \int_0^1 h_j(p) dp + (3 - \gamma) h_j \left( \frac{1}{\gamma} \right) \right).$$

Now we present our main result for the FLPLP which follows the approach of Li [18].

**Theorem 10.** *Algorithm 4 produces a solution with expected cost*

$$\mathbb{E}[F + C + P] \leq 1.5148(F^* + C^* + P^*),$$

*implying that Algorithm 4 is a 1.5148-approximation algorithm.*

**Acknowledgements.** The research of the second author is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) grant 283106. The third author's research is supported by the National Basic Research Program of China (No. 2010CB732501). The fourth author's research is supported by NSF of China (No. 11071268) and Scientific Research Common Program of Beijing Municipal Commission of Education (No. KM201210005033).

## References

1. Aardal, K.I., Chudak, F.A., Shmoys, D.B.: A 3-approximation algorithm for the  $k$ -level uncapacitated facility location problem. Information Processing Letters 72, 161–167 (1999)

2. Ageev, A., Ye, Y., Zhang, J.: Improved combinatorial approximation algorithms for the  $k$ -level facility location problem. *SIAM Journal on Discrete Mathematics* 18, 207–217 (2003)
3. Byrka, J., Aardal, K.I.: An optimal bi-factor approximation algorithm for the metric uncapacitated facility location problem. *SIAM Journal on Computing* 39, 2212–2231 (2010)
4. Charikar, M., Guha, S.: Improved combinatorial algorithms for facility location and  $k$ -median problems. In: *Proceedings of FOCS*, pp. 378–388 (1999)
5. Charikar, M., Khuller, S., Mount, D.M., Naraasimban, G.: Algorithms for facility location problems with outliers. In: *Proceedings of SODA*, pp. 642–651 (2001)
6. Chen, X., Chen, B.: Approximation algorithms for soft-capacitated facility location in capacitated network design. *Algorithmica* 53, 263–297 (2007)
7. Chudak, F.A., Nagano, K.: Efficient solutions to relaxations of combinatorial problems with submodular penalty via the Lovász extension and non-smooth convex optimization. In: *Proceedings of SODA*, pp. 79–88 (2007)
8. Chudak, F.A., Shmoys, D.B.: Improved approximation algorithms for the uncapacitated facility location problem. *SIAM Journal on Computing* 33, 1–25 (2003)
9. Du, D., Lu, R., Xu, D.: A primal-dual approximation algorithm for the facility location problem with submodular penalty. *Algorithmica* 63, 191–200 (2012)
10. Fujishige, S.: Submodular Functions and Optimization, 2nd edn. *Annals of Discrete Mathematics*, vol. 58. Elsevier (2005)
11. Geunes, J., Levi, R., Romeijn, H.E., Shmoys, D.B.: Approximation algorithms for supply chain planning and logistics problems with market choice. *Mathematical Programming* 130, 85–106 (2011)
12. Guha, S., Khuller, S.: Greedy strike back: improved facility location algorithms. *Journal of Algorithms* 31, 228–248 (1999)
13. Hayrapetyan, A., Swamy, C., Tardös, E.: Network design for information networks. In: *Proceedings of SODA*, pp. 933–942 (2005)
14. Jain, K., Mahdian, M., Markakis, E., Saberi, A., Vazirani, V.V.: Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM* 50, 795–824 (2003)
15. Jain, K., Mahdian, M., Saberi, A.: A new greedy approach for facility location problems. In: *Proceedings of STOC*, pp. 731–740 (2002)
16. Jain, K., Vazirani, V.V.: Approximation algorithms for metric facility location and  $k$ -median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM* 48, 274–296 (2001)
17. Korupolu, M.R., Plaxton, C.G., Rajaraman, R.: Analysis of a local search heuristic for facility location problems. In: *Proceedings of SODA*, pp. 1–10 (1998)
18. Li, S.: A 1.488 Approximation algorithm for the uncapacitated facility location problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) *ICALP 2011, Part II. LNCS*, vol. 6756, pp. 77–88. Springer, Heidelberg (2011)
19. Mahdian, M.: Facility location and the analysis of algorithms through factor-revealing programs. Ph. D. thesis, MIT, Cambridge, MA (2004)
20. Mahdian, M., Ye, Y., Zhang, J.: Improved approximation algorithms for metric facility location problems. *SIAM Journal on Computing* 36, 411–432 (2006)
21. Shmoys, D.B., Swamy, C.: An approximation scheme for stochastic linear programming and its application to stochastic integer programs. *Journal of the ACM* 53, 978–1012 (2006)
22. Shmoys, D.B., Tardös, E., Aardal, K.I.: Approximation algorithms for facility location problems. In: *Proceedings of STOC*, pp. 265–274 (1997)

23. Shu, J., Teo, C.P., Shen, Z.J.: Max: Stochastic transportation-inventory network design problem. *Operations Research* 53, 48–60 (2005)
24. Sviridenko, M.: An improved approximation algorithm for the metric uncapacitated facility location problem. In: Cook, W.J., Schulz, A.S. (eds.) *IPCO 2002*. LNCS, vol. 2337, pp. 240–257. Springer, Heidelberg (2002)
25. Vygen, J.: Approximation algorithms for facility location problems (Lecture Notes). Report No. 05950-OR, Research Institute for Discrete Mathematics, University of Bonn (2005), <http://www.or.uni-bonn.de/vygen/f1.pdf>
26. Xu, G., Xu, J.: An LP rounding algorithm for approximating uncapacitated facility location problem with penalty. *Information Processing Letters* 94, 119–123 (2005)
27. Xu, G., Xu, J.: An improved approximation algorithm for uncapacitated facility location problem with penalty. *Journal of Combinatorial Optimization* 17, 424–436 (2008)
28. Ye, Y., Zhang, J.: An approximation algorithm for the dynamic facility location problem. In: *Combinatorial Optimization in Communication Networks*, pp. 623–637. Kluwer Academic Publishers (2005)
29. Zhang, J.: Approximating the two-level facility location problem via a quasi-greedy approach. *Mathematical Programming* 108, 159–176 (2006)
30. Zhang, J., Chen, B., Ye, Y.: A multiechange local search algorithm for the capacitated facility location problem. *Mathematics of Operations Research* 30, 389–403 (2005)

# An Improved Semidefinite Programming Hierarchies Rounding Approximation Algorithm for Maximum Graph Bisection Problems

Chenchen Wu<sup>1</sup>, Donglei Du<sup>2</sup>, and Dachuan Xu<sup>3,\*</sup>

<sup>1</sup> School of Mathematical Sciences, Nankai University, Tianjin 300071, P.R. China

<sup>2</sup> Faculty of Business Administration, University of New Brunswick, NB Canada  
Fredericton E3B 9Y2

<sup>3</sup> Department of Applied Mathematics, Beijing University of Technology,  
100 Pingleyuan, Chaoyang District, Beijing 100124, P.R. China  
[xudc@bjut.edu.cn](mailto:xudc@bjut.edu.cn)

**Abstract.** We present a unified semidefinite programming hierarchies rounding approximation algorithm for a class of maximum graph bisection problems with improved approximation ratios.

**Keywords:** Semidefinite programming hierarchies, approximation algorithm, graph bisection problems.

## 1 Introduction

The semidefinite programming (SDP) hyperplane rounding technique was initialized in the seminal paper of Goemans and Williamson [4], and has been applied to design approximation algorithms for many combinatorial optimization problems, including graph bisection problems [2,3,4,5,6,9,10]. The main interest of this work is to provide improved approximation ratios for these maximum graph bisection problems by a new variant of the SDP rounding, namely, the semidefinite programming hierarchies rounding (SDPH), which was recently developed by Lasserre [7] and has become a powerful tool in designing approximation algorithms for the maximum bisection cut problem [1,8]. The SDPH is also called the Lasserre SDP relaxation, which will be used interchangeably in this paper.

Given an undirected graph  $G = (V, E)$  with nonnegative edge weights  $w_{ij}$  ( $(i, j) \in E$ ) and even number of nodes  $n = |V|$ , we consider six types of maximum graph bisection problems, where the first four are defined on undirected graph, and the last two on directed graph. A *CUT*  $(S, \bar{S})$  in a undirected graph contains all edges between  $S \subseteq V$  and  $\bar{S} = V \setminus S$ . A directed *cut*  $(S, \bar{S})$  contains all directed edges from  $S \subseteq V$  to  $\bar{S} = V \setminus S$ . A  $k$ -*CUT* is a cut  $(S, \bar{S})$  such that  $|S| = k$ .

- (1) MAX- $\frac{n}{2}$ -CUT: find a  $\frac{n}{2}$ -CUT  $(S, \bar{S})$  to maximize the total weight of cut edges, namely  $\sum_{(i,j) \in (S, \bar{S})} w_{ij}$ ;

---

\* Corresponding author.

- (2) MAX- $\frac{n}{2}$ -UNCUT: find a  $\frac{n}{2}$ -CUT to maximize the total weight of edges that do not cross the cut, namely  $\sum_{(i,j) \notin (S, \bar{S})} w_{ij}$ ;
- (3) MAX- $\frac{n}{2}$ -DENSE-SUBGRAPH: find a vertex  $S \subseteq V$  with  $|S| = \frac{n}{2}$  to maximize the total weight of edges within one component, namely  $\sum_{i \in S} \sum_{j \in V} w_{ij}$ ;
- (4) MAX- $\frac{n}{2}$ -VERTEX-COVER: find a vertex  $S \subseteq V$  with  $|S| = \frac{n}{2}$  to maximize the total weight of edges touching this set, namely  $\sum_{i \in S, j \in V} w_{ij}$ ;
- (5) MAX- $\frac{n}{2}$ -DIRECTED-CUT: find a directed  $\frac{n}{2}$ -CUT  $(S, \bar{S})$  to maximize the total weight of cut edges, namely  $\sum_{(i,j) \in (S, \bar{S})} w_{ij}$ ;
- (6) MAX- $\frac{n}{2}$ -DIRECTED-UNCUT: find a directed  $\frac{n}{2}$ -CUT to maximize the total weight of edges that do not cross the cut, namely  $\sum_{(i,j) \notin (S, \bar{S})} w_{ij}$ .

Table 1 below summarizes the best known approximation ratios for these problems, where the last column contains results obtained via the standard SDP rounding technique, and the second column contains results obtained by the semidefinite programming hierarchies (SDPH) rounding technique.

**Table 1.** Approximation ratios

Problem	Ratio (SDPH)	Ratio (SDP)
MAX- $\frac{n}{2}$ -CUT	0.8776 [1]	0.7028 [2]
MAX- $\frac{n}{2}$ -UNCUT	0.8776 (this work)	0.6436 [5]
MAX- $\frac{n}{2}$ -DENSE-SUBGRAPH	0.8731 (this work)	0.6236 [9]
MAX- $\frac{n}{2}$ -VERTEX-COVER	0.9401 (this work)	0.8452 [5]
MAX- $\frac{n}{2}$ -DIRECTED-CUT	0.8731 (this work)	0.6458 [9]
MAX- $\frac{n}{2}$ -DIRECTED-UNCUT	0.9401 (this work)	0.8118 [5]

The main technique in obtaining the improved ratios (Column 2 in Table 1) in this work involves integrating the SDPH rounding technique [1,8] and the size adjusting technique [6]. The main contributions of our paper are summarized as follows:

- (i) We present a unified SDPH rounding approximation algorithm for six maximum graph bisection problems.
- (ii) Under the above algorithmic framework, we show that the approximation ratios of MAX- $\frac{n}{2}$ -CUT, MAX- $\frac{n}{2}$ -DENSE-SUBGRAPH, and MAX- $\frac{n}{2}$ -VERTEX-COVER are equal to those of MAX- $\frac{n}{2}$ -UNCUT, MAX- $\frac{n}{2}$ -DIRECTED-CUT, and MAX- $\frac{n}{2}$ -DIRECTED-UNCUT, respectively.

In the remainder of this paper, we present some preliminaries in Section 2. The SDPH rounding algorithm and its analysis are given in Sections 3 and 4 respectively. Finally, we give some concluding remarks in Section 5.

## 2 Preliminaries

In order to give a unified mathematical program for the maximum graph bisection problems, we adopt the following notations (cf. [5,6]) which are listed in Table 2.

**Table 2.** Notations

Problem	$c_0$	$c_1$	$c_2$	$c_3$
MAX- $\frac{n}{2}$ -CUT	1/2	0	0	-1/2
MAX- $\frac{n}{2}$ -UNCUT	1/2	0	0	1/2
MAX- $\frac{n}{2}$ -DENSE-SUBGRAPH	1/4	1/4	1/4	1/4
MAX- $\frac{n}{2}$ -VERTEX-COVER	3/4	1/4	1/4	-1/4
MAX- $\frac{n}{2}$ -DIRECTED-CUT	1/4	1/4	-1/4	-1/4
MAX- $\frac{n}{2}$ -DIRECTED-UNCUT	3/4	-1/4	1/4	1/4

These maximum graph bisection problems can be unified as the following binary integer quadratic program:

$$\begin{aligned} OPT := \max & \sum_{(i,j) \in E} w_{ij} (c_0 + c_1 x_0 x_i + c_2 x_0 x_j + c_3 x_i x_j) \\ \text{s. t. } & \sum_{i \in V} x_0 x_i = 0, \\ & x_i \in \{-1, 1\}, \quad \forall i \in V, \end{aligned}$$

along with its SDP relaxation:

$$\begin{aligned} \max & \sum_{(i,j) \in E} w_{ij} (c_0 + c_1 \langle v_0, v_i \rangle + c_2 \langle v_0, v_j \rangle + c_3 \langle v_i, v_j \rangle) \\ \text{s. t. } & \left\langle v_0, \sum_{i \in V} v_i \right\rangle = 0, \\ & \left\langle \sum_{i \in V} v_i, \sum_{i \in V} v_i \right\rangle = 0, \\ & \langle v_i, v_i \rangle = 1, \quad \forall i \in V \cup \{0\}. \end{aligned}$$

In order to introduce the  $\ell$ th-round Lasserre SDP relaxation, we present the following equivalent binary integer program involving  $\prod_{i \in S} x_i$  for any small set  $S \in \mathcal{S} := \{S : S \subseteq V, |S| \leq \ell\}$ :

$$\begin{aligned} \max & \sum_{(i,j) \in E} w_{ij} (c_0 + c_1 x_0 x_i + c_2 x_0 x_j + c_3 x_i x_j) \\ \text{s. t. } & x_0 \sum_{i \in V} \left( \prod_{j \in S} x_j \right) x_i = 0, \quad S \in \mathcal{S}, \\ & \prod_{i \in S_1} x_i \prod_{j \in S_2} x_j = \prod_{i \in S_3} x_i \prod_{j \in S_4} x_j, \quad \forall S_1, S_2, S_3, S_4 \in \mathcal{S}, \text{ such that } S_1 \Delta S_2 = S_3 \Delta S_4, \\ & x_i \in \{-1, 1\}, \quad \forall i \in V \cup \{0\}, \end{aligned}$$

in which the symbol  $\Delta$  denotes the operation of symmetric difference. Introduce a variable  $v_S$  which simulates  $\prod_{i \in S} x_i$  for any small set  $S \in \mathcal{S}$ . Then we get the  $\ell$ th-round Lasserre SDP relaxation:

$$\begin{aligned}
\max \quad & \sum_{(i,j) \in E} w_{ij} (c_0 + c_1 \langle v_0, v_i \rangle + c_2 \langle v_0, v_j \rangle + c_3 \langle v_i, v_j \rangle) \\
\text{s. t. } & \left\langle v_\emptyset, \sum_{i \in V} v_{S \Delta \{i\}} \right\rangle = 0, \quad \forall S \in \mathcal{S}, \\
& \langle v_{S_1}, v_{S_2} \rangle = \langle v_{S_3}, v_{S_4} \rangle, \forall S_1, S_2, S_3, S_4 \in \mathcal{S}, \text{ such that } S_1 \Delta S_2 = S_3 \Delta S_4, \\
& \langle v_\emptyset, v_\emptyset \rangle = 1.
\end{aligned}$$

In the above SDPH, we also use  $v_0$  for  $v_\emptyset$  alternatively.

Throughout the paper, denote  $\Phi(t)$  as the cumulative distribution function of the standard normal random variable, and let  $\Phi^{-1}$  be the inverse function of  $\Phi$ :

$$\Phi(t) := \int_{-\infty}^t \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{s^2}{2}\right\} ds, \quad \Phi^{-1} : [0, 1] \longrightarrow [-\infty, \infty].$$

### 3 Algorithm

Now we are ready to present the unified SDPH rounding algorithm for the maximum graph bisection problems.

#### Algorithm 1 (Algorithm SDPH-GB)

##### Step 0 (Initialization)

$\theta \in [0, 1]$ ,  $\varepsilon > 0$ , and  $t := (20 \lceil 1/\varepsilon \rceil)^{12}$ .

##### Step 1 (SDPH-GB solving)

Solve the  $(t+1)$ th-round Lasserre SDP relaxation to obtain an optimal solution  $\{v_S : |S| \leq t+1\}$ .

##### Step 2 (Sampling by conditioning)

Repeat Steps 2.0-2.4 below until there exists a set of vectors  $v'_0, v'_1, \dots, v'_n$  such that

$$\begin{aligned}
\sum_{(i,j) \in E} w_{ij} (c_0 + c_1 \langle v'_0, v'_i \rangle + c_2 \langle v'_0, v'_j \rangle + c_3 \langle v'_i, v'_j \rangle) &\geq OPT - 9\varepsilon/20, \\
\sum_{i,j \in V} |\langle w'_i, w'_j \rangle| &\leq (\varepsilon/20)^5,
\end{aligned}$$

where

$$w'_i = v'_i - \langle v'_0, v'_i \rangle v'_0.$$

**Step 2.0**  $v_S^{t+1} := v_S$  for all  $|S| \leq t+1$ , and  $k := 1$ .

**Step 2.1** Pick an index  $j_k \in V$  at random.

**Step 2.2** Sample an assignment  $a \in \{-1, 1\}$  for index  $j_k$  from its marginal distribution.

**Step 2.3** Condition distribution on this assignment.  $v_S^{t+1-k} := v_{S \cup \{j_k\}}^{t+2-k}$  for all  $|S| \leq t+1-k$ . Then  $\{v_S^{t+1-k} : |S| \leq t+1-k\}$  is an optimal solution for the  $(t+1-k)$ th-round Lasserre SDP relaxation.

**Step 2.4** If  $k = t$ , stop; otherwise,  $k := k + 1$ , go to Step 2.1.

**Step 3 (Randomized rounding)**

**Step 3.1** Define

$$v_0 := v'_0,$$

$$v_i := \begin{cases} v'_i, & \text{if } \|w'_i\|_2 \geq t^{-1/12}; \\ v'_i - w'_i + w^*_i, & \text{otherwise,} \end{cases} \quad i = 1, \dots, n,$$

where  $w^*_i$  is a new vector of length  $\|w'_i\|_2$  and orthogonal to all other vectors.

**Step 3.2** For  $i = 1, \dots, n$ ,

$$\mu_i := \langle v_0, v_i \rangle,$$

$$w_i := v_i - \mu_i v_0,$$

$$\bar{w}_i := \begin{cases} w_i / \|w_i\|_2 & \text{if } \|w_i\|_2 \neq 0; \\ \text{a unit vector orthogonal to all other vectors, if } \|w_i\|_2 = 0. \end{cases}$$

**Step 3.3** Pick a standard  $n$ -dimensional Gaussian random vector  $g$ . Initialize  $\tilde{x}_0 := 1$ . For all  $i = 1, \dots, n$ ,

$$\tilde{x}_i := \begin{cases} -1, & \text{if } \langle \bar{w}_i, g \rangle < \Phi^{-1} \left( \frac{1 - \theta \mu_i}{2} \right); \\ 1, & \text{otherwise.} \end{cases}$$

**Step 4 (Size adjusting [6])** Let  $\tilde{S} := \{i \in V : \tilde{x}_i = 1\}$ . If  $|\tilde{S}| \neq \frac{n}{2}$ , greedily adjust the size to  $\frac{n}{2}$ . Let us denote the adjusted binary solution  $\{\hat{x}_i\}$  with  $\hat{x}_0 = 1$  and the corresponding set  $\hat{S} := \{i \in V : \hat{x}_i = 1\}$ .

For any  $i, j \in V$ , denote  $\rho_{ij} := \langle v_i, v_j \rangle$ . In the following proof, we need to consider  $\Pr\{\tilde{x}_i = \tilde{x}_j\}$  for any  $i, j \in V$ , which motivates us to focus on the probability distributions of the two variables  $\langle \bar{w}_i, g \rangle$  and  $\langle \bar{w}_j, g \rangle$ , where  $g$  is the standard  $n$ -dimensional Gaussian random vector. Evidently  $\langle \bar{w}_i, g \rangle$  and  $\langle \bar{w}_j, g \rangle$  are jointly normal random variables with mean 0 and covariance matrix

$$\begin{pmatrix} \langle \bar{w}_i, \bar{w}_i \rangle & \langle \bar{w}_i, \bar{w}_j \rangle \\ \langle \bar{w}_i, \bar{w}_j \rangle & \langle \bar{w}_j, \bar{w}_j \rangle \end{pmatrix} = \begin{pmatrix} 1 & h_{ij} \\ h_{ij} & 1 \end{pmatrix},$$

where

$$h_{ij} = h(\mu_i, \mu_j, \rho_{ij}) := \begin{cases} 0, & \text{if } \mu_i = \pm 1, \text{ or } \mu_j = \pm 1; \\ \frac{\rho_{ij} - \mu_i \mu_j}{\sqrt{(1 - \mu_i^2)(1 - \mu_j^2)}}, & \text{otherwise.} \end{cases}$$

Denote the corresponding density function

$$p(s_i, s_j; h_{ij}) := \frac{1}{2\pi\sqrt{1 - h_{ij}^2}} \exp \left\{ -\frac{1}{2(1 - h_{ij}^2)} [s_i^2 - 2h_{ij}s_i s_j + s_j^2] \right\}.$$

It follows from Lemma 3.4 in [1] that

$$\Pr\{\tilde{x}_i = \tilde{x}_j\} = \frac{\theta}{2}(\mu_i + \mu_j) + 2 \int_{-\infty}^{\Phi^{-1}\left(\frac{1-\theta\mu_i}{2}\right)} \int_{-\infty}^{\Phi^{-1}\left(\frac{1-\theta\mu_j}{2}\right)} p(s_i, s_j; h_{ij}) ds_i ds_j. \quad (1)$$

Now we consider Step 4 in Algorithm 1 (the size adjusting). Denote

$$\begin{aligned} w(\tilde{S}) &:= \sum_{(i,j) \in E} (c_0 + c_1 \tilde{x}_0 \tilde{x}_i + c_2 \tilde{x}_0 \tilde{x}_j + c_3 \tilde{x}_i \tilde{x}_j), \\ w(\hat{S}) &:= \sum_{(i,j) \in E} (c_0 + c_1 \hat{x}_0 \hat{x}_i + c_2 \hat{x}_0 \hat{x}_j + c_3 \hat{x}_i \hat{x}_j). \end{aligned}$$

The following lemma comes from [6] which shows the relationship between  $w(\tilde{S})$  and  $w(\hat{S})$ .

**Lemma 1.** ([6]) *The size adjusting of Algorithm 1 has the following properties with respect to the six types of maximum graph bisection problems.*

– MAX- $\frac{n}{2}$ -CUT.

$$w(\hat{S}) \geq \begin{cases} \frac{n/2}{|\tilde{S}|} w(\tilde{S}), & \text{if } |\tilde{S}| \geq \frac{n}{2}; \\ \frac{n/2}{n - |\tilde{S}|} w(\tilde{S}), & \text{otherwise.} \end{cases}$$

– MAX- $\frac{n}{2}$ -UNCUT.

$$w(\hat{S}) \geq \begin{cases} \frac{\frac{n}{2}(\frac{n}{2} - 1)}{|\tilde{S}|(|\tilde{S}| - 1)} w(\tilde{S}), & \text{if } |\tilde{S}| \geq \frac{n}{2}; \\ \frac{\frac{n}{2}(\frac{n}{2} - 1)}{(n - |\tilde{S}|)(n - |\tilde{S}| - 1)} w(\tilde{S}), & \text{otherwise.} \end{cases}$$

– MAX- $\frac{n}{2}$ -DENSE-SUBGRAPH.

$$w(\hat{S}) \geq \begin{cases} \frac{\frac{n}{2}(\frac{n}{2} - 1)}{|\tilde{S}|(|\tilde{S}| - 1)} w(\tilde{S}), & \text{if } |\tilde{S}| \geq \frac{n}{2}; \\ w(\tilde{S}), & \text{otherwise.} \end{cases}$$

– MAX- $\frac{n}{2}$ -VERTEX-COVER.

$$w(\hat{S}) \geq \begin{cases} \frac{n/2}{|\tilde{S}|} w(\tilde{S}), & \text{if } |\tilde{S}| \geq \frac{n}{2}; \\ w(\tilde{S}), & \text{otherwise.} \end{cases}$$

– MAX- $\frac{n}{2}$ -DIRECTED-CUT.

$$w(\hat{S}) \geq \begin{cases} \frac{n/2}{|\tilde{S}|} w(\tilde{S}), & \text{if } |\tilde{S}| \geq \frac{n}{2}; \\ \frac{n/2}{n - |\tilde{S}|} w(\tilde{S}), & \text{otherwise.} \end{cases}$$

– MAX- $\frac{n}{2}$ -DIRECTED-UNCUT.

$$w(\hat{S}) \geq \begin{cases} \frac{\frac{n}{2}(\frac{n}{2}-1)}{|\tilde{S}|(|\tilde{S}|-1)} w(\tilde{S}), & \text{if } |\tilde{S}| \geq \frac{n}{2}; \\ \frac{\frac{n}{2}(\frac{n}{2}-1)}{(n-|\tilde{S}|)(n-|\tilde{S}|-1)} w(\tilde{S}), & \text{otherwise.} \end{cases}$$

## 4 Analysis

First, Algorithm 1 is well-defined thanks to the following lemma.

**Lemma 2.** ([1]) *The process of sampling by conditioning in Algorithm 1 is implemented in (expected)  $60\lceil 1/\varepsilon \rceil$  times. Moreover, the vectors  $v_0, \dots, v_n$  produced by Algorithm 1 satisfies the following properties:*

- (a)  $\sum_{(i,j) \in E} w_{ij} (c_0 + c_1 \langle v_0, v_i \rangle + c_2 \langle v_0, v_j \rangle + c_3 \langle v_i, v_j \rangle) \geq OPT - \varepsilon/2$ ;
- (b)  $\sum_{i \in V} \langle v_0, v_i \rangle = 0$ ;
- (c) We have the following triangle inequalities:

$$\begin{aligned} \mu_i + \mu_j + \rho_{ij} &\geq -1, & \mu_i - \mu_j - \rho_{ij} &\geq -1, \\ -\mu_i + \mu_j - \rho_{ij} &\geq -1, & -\mu_i - \mu_j + \rho_{ij} &\geq -1; \end{aligned}$$

$$(d) \sum_{i,j \in V} |\langle \bar{w}_i, \bar{w}_j \rangle| \leq (\varepsilon/20)^3.$$

The above lemma implies that the objective value of  $\{v_i\}$  from randomized rounding (in polynomial time) is close to the optimal value, which can thus be replaced by that of  $\{v_i\}$  in the approximation ratio analysis. Moreover, the last conclusion (d) in the above lemma implies that the differences between the cardinality of  $\tilde{S}$  and  $V \setminus \tilde{S}$  is small, resulting in Lemma 7 shortly.

Second, we estimate the first two moments of solution  $\{\tilde{x}\}$ .

**Lemma 3.** *For any  $i, j \in V$ , we have  $E[\tilde{x}_i] = \theta\mu_i$  and*

$$E[\tilde{x}_i \tilde{x}_j] = f(\mu_i, \mu_j, \rho_{ij}),$$

where

$$f(\mu_i, \mu_j, \rho_{ij}) := \theta(\mu_i + \mu_j) - 1 + 4 \int_{-\infty}^{\Phi^{-1}\left(\frac{1-\theta\mu_i}{2}\right)} \int_{-\infty}^{\Phi^{-1}\left(\frac{1-\theta\mu_j}{2}\right)} p(s_i, s_j; h_{ij}) ds_i ds_j.$$

*Proof.* From the definition of  $\{\tilde{x}\}$ , we have

$$\begin{aligned} E[\tilde{x}_i] &= \Pr\{\tilde{x}_i = 1\} - \Pr\{\tilde{x}_i = -1\} = 1 - 2\Pr\{\tilde{x}_i = -1\} \\ &= 1 - 2\Pr\left\{\langle \bar{w}_i, g \rangle < \Phi^{-1}\left(\frac{1-\theta\mu_i}{2}\right)\right\} = 1 - 2\frac{1-\theta\mu_i}{2} = \theta\mu_i. \end{aligned}$$

From (1), we have

$$\begin{aligned}
E[\tilde{x}_i \tilde{x}_j] &= \Pr\{\tilde{x}_i \tilde{x}_j = 1\} - \Pr\{\tilde{x}_i \tilde{x}_j = -1\} = 2\Pr\{\tilde{x}_i \tilde{x}_j = 1\} - 1 \\
&= 2\Pr\{\tilde{x}_i = \tilde{x}_j\} - 1 \\
&= \theta(\mu_i + \mu_j) - 1 + 4 \int_{-\infty}^{\Phi^{-1}\left(\frac{1-\theta\mu_i}{2}\right)} \int_{-\infty}^{\Phi^{-1}\left(\frac{1-\theta\mu_j}{2}\right)} p(s_i, s_j; h_{ij}) ds_i ds_j \\
&= f(\mu_i, \mu_j, \rho_{ij}).
\end{aligned}$$

□

Denote

$$\mathcal{F} := \left\{ (\mu_1, \mu_2, \rho) \in [-1, 1]^3 : \begin{pmatrix} 1 & \mu_1 & \mu_2 \\ \mu_1 & 1 & \rho \\ \mu_2 & \rho & 1 \end{pmatrix} \succeq 0, \begin{array}{l} \mu_1 + \mu_2 + \rho \geq -1 \\ \mu_1 - \mu_2 - \rho \geq -1 \\ -\mu_1 + \mu_2 - \rho \geq -1 \\ -\mu_1 - \mu_2 + \rho \geq -1 \end{array} \right\},$$

and

$$\alpha(\theta; c_0, c_1, c_2, c_3) := \min_{(\mu_1, \mu_2, \rho) \in \mathcal{F}} \frac{c_0 + c_1\theta\mu_1 + c_2\theta\mu_2 + c_3f(\mu_1, \mu_2, \rho)}{c_0 + c_1\mu_1 + c_2\mu_2 + c_3\rho}.$$

To simply the notions, let  $\alpha(\theta) := \alpha(\theta; c_0, c_1, c_2, c_3)$ . Then we have

**Lemma 4.**

$$E[w(\tilde{S})] \geq \alpha(\theta) \sum_{(i,j) \in E} (c_0 + c_1\langle v_0, v_i \rangle + c_2\langle v_0, v_j \rangle + c_3\langle v_i, v_j \rangle).$$

*Proof.* It follows from Algorithm 1, Lemma 3 and the definitions of  $\mathcal{F}$  and  $\alpha(\theta)$  that

$$\begin{aligned}
&E[w(\tilde{S})] \\
&= \sum_{(i,j) \in E} (c_0 + c_1E[\tilde{x}_i] + c_2E[\tilde{x}_j] + c_3E[\tilde{x}_i \tilde{x}_j]) \\
&= \sum_{(i,j) \in E} (c_0 + c_1\theta\mu_i + c_2\theta\mu_j + c_3f(\mu_i, \mu_j, \rho_{ij})) \\
&= \sum_{(i,j) \in E} \frac{c_0 + c_1\theta\mu_i + c_2\theta\mu_j + c_3f(\mu_i, \mu_j, \rho_{ij})}{c_0 + c_1\mu_i + c_2\mu_j + c_3\rho_{ij}} (c_0 + c_1\mu_i + c_2\mu_j + c_3\rho_{ij}) \\
&\geq \alpha(\theta) \sum_{(i,j) \in E} (c_0 + c_1\langle v_0, v_i \rangle + c_2\langle v_0, v_j \rangle + c_3\langle v_i, v_j \rangle).
\end{aligned}$$

□

Third, we present some equivalent properties for the ratios  $\alpha(\theta)$ .

**Lemma 5.** For any  $(\mu_1, \mu_2, \rho) \in \mathcal{F}$ , we have

- (a)  $f(\mu_1, -\mu_2, -\rho) = -f(\mu_1, \mu_2, \rho)$ ;
- (b)  $f(-\mu_1, \mu_2, -\rho) = f(\mu_1, \mu_2, \rho)$ .

*Proof.* Since the proof of (b) is analogous to (a), we only prove (a) below. Denote that  $h_0 = h(\mu_1, \mu_2, \rho)$ . By the definition of  $h(\cdot)$ , we have  $h(\mu_1, -\mu_2, -\rho) = -h(\mu_1, \mu_2, \rho)$ . Together with the definition of  $p(\cdot)$ , we have

$$\begin{aligned} & p(s_i, s_j; h(\mu_1, -\mu_2, -\rho)) \\ &= \frac{1}{2\pi\sqrt{1-h(\mu_1, -\mu_2, -\rho)^2}} \exp \left\{ -\frac{1}{2(1-h(\mu_1, -\mu_2, -\rho)^2)} [s_i^2 - 2h(\mu_1, -\mu_2, -\rho)s_i s_j + s_j^2] \right\} \\ &= \frac{1}{2\pi\sqrt{1-h_0^2}} \exp \left\{ -\frac{1}{2(1-h_0^2)} [s_i^2 + 2h_0 s_i s_j + s_j^2] \right\} \end{aligned}$$

Noting that  $\Phi^{-1}(\frac{1}{2} + x) = -\Phi^{-1}(\frac{1}{2} - x)$  and  $\int_{-\infty}^{+\infty} p(s_i, s_j; h_0) ds_j = 1$  for any fix  $s_i$ , we have

$$\begin{aligned} & \int_{-\infty}^{\Phi^{-1}\left(\frac{1-\theta\mu_1}{2}\right)} \int_{-\infty}^{\Phi^{-1}\left(\frac{1+\theta\mu_2}{2}\right)} p(s_i, s_j; h(\mu_1, -\mu_2, -\rho)) ds_i ds_j \\ &= \int_{-\infty}^{\Phi^{-1}\left(\frac{1-\theta\mu_1}{2}\right)} \int_{-\infty}^{-\Phi^{-1}\left(\frac{1-\theta\mu_2}{2}\right)} \frac{1}{2\pi\sqrt{1-h_0^2}} \exp \left\{ -\frac{1}{2(1-h_0^2)} [s_i^2 + 2h_0 s_i s_j + s_j^2] \right\} ds_i ds_j \\ &= \int_{-\infty}^{\Phi^{-1}\left(\frac{1-\theta\mu_1}{2}\right)} \int_{\Phi^{-1}\left(\frac{1-\theta\mu_2}{2}\right)}^{+\infty} \frac{1}{2\pi\sqrt{1-h_0^2}} \exp \left\{ -\frac{1}{2(1-h_0^2)} [s_i^2 - 2h_0 s_i s_j + s_j^2] \right\} ds_i ds_j \\ &= \int_{-\infty}^{\Phi^{-1}\left(\frac{1-\theta\mu_1}{2}\right)} \int_{\Phi^{-1}\left(\frac{1-\theta\mu_2}{2}\right)}^{+\infty} p(s_i, s_j; h_0) ds_i ds_j \\ &= \int_{-\infty}^{\Phi^{-1}\left(\frac{1-\theta\mu_1}{2}\right)} \left( 1 - \int_{-\infty}^{\Phi^{-1}\left(\frac{1-\theta\mu_2}{2}\right)} p(s_i, s_j; h_0) ds_j \right) ds_i \\ &= \frac{1-\theta\mu_1}{2} - \int_{-\infty}^{\Phi^{-1}\left(\frac{1-\theta\mu_1}{2}\right)} \int_{-\infty}^{\Phi^{-1}\left(\frac{1-\theta\mu_2}{2}\right)} p(s_i, s_j; h_0) ds_i ds_j. \end{aligned}$$

The above equality together with the definition of  $f$  imply that

$$\begin{aligned} & f(\mu_1, -\mu_2, -\rho) \\ &= \theta(\mu_1 - \mu_2) - 1 + 4 \int_{-\infty}^{\Phi^{-1}\left(\frac{1-\theta\mu_1}{2}\right)} \int_{-\infty}^{\Phi^{-1}\left(\frac{1+\theta\mu_2}{2}\right)} p(s_i, s_j; h(\mu_1, -\mu_2, -\rho)) ds_i ds_j \\ &= \theta(\mu_1 - \mu_2) - 1 + 4 \frac{1-\theta\mu_1}{2} - 4 \int_{-\infty}^{\Phi^{-1}\left(\frac{1-\theta\mu_1}{2}\right)} \int_{-\infty}^{\Phi^{-1}\left(\frac{1-\theta\mu_2}{2}\right)} p(s_i, s_j; h_0) ds_i ds_j \\ &= -\theta(\mu_1 + \mu_2) + 1 - 4 \int_{-\infty}^{\Phi^{-1}\left(\frac{1-\theta\mu_1}{2}\right)} \int_{-\infty}^{\Phi^{-1}\left(\frac{1-\theta\mu_2}{2}\right)} p(s_i, s_j; h(\mu_1, \mu_2, \rho)) ds_i ds_j \\ &= -f(\mu_1, \mu_2, \rho). \end{aligned}$$

□

**Lemma 6.** *The ratios  $\alpha(\theta)$  for different maximum graph bisection problems have the following properties:*

- (a)  $\alpha\left(\theta; \frac{1}{2}, 0, 0, \frac{1}{2}\right) = \alpha\left(\theta; \frac{1}{2}, 0, 0, -\frac{1}{2}\right);$
- (b)  $\alpha\left(\theta; \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right) = \alpha\left(\theta; \frac{1}{4}, \frac{1}{4}, -\frac{1}{4}, -\frac{1}{4}\right);$
- (c)  $\alpha\left(\theta; \frac{3}{4}, \frac{1}{4}, \frac{1}{4}, -\frac{1}{4}\right) = \alpha\left(\theta; \frac{3}{4}, -\frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right).$

*Proof.* By Lemma 5 and the definition of  $\alpha$ , we have the following relations.

(a)

$$\begin{aligned} \alpha\left(\theta; \frac{1}{2}, 0, 0, -\frac{1}{2}\right) &= \min_{(\mu_1, \mu_2, \rho) \in \mathcal{F}} \frac{1 - f(\mu_1, \mu_2, \rho)}{1 - \rho} \\ &= \min_{(\mu_1, \mu_2, \rho) \in \mathcal{F}} \frac{1 - f(\mu_1, -\mu_2, -\rho)}{1 + \rho} \\ &= \min_{(\mu_1, \mu_2, \rho) \in \mathcal{F}} \frac{1 + f(\mu_1, \mu_2, \rho)}{1 + \rho} \\ &= \alpha\left(\theta; \frac{1}{2}, 0, 0, \frac{1}{2}\right). \end{aligned}$$

(b)

$$\begin{aligned} \alpha\left(\theta; \frac{1}{4}, \frac{1}{4}, -\frac{1}{4}, -\frac{1}{4}\right) &= \min_{(\mu_1, \mu_2, \rho) \in \mathcal{F}} \frac{1 + \theta\mu_1 - \theta\mu_2 - f(\mu_1, \mu_2, \rho)}{1 + \mu_1 - \mu_2 - \rho} \\ &= \min_{(\mu_1, \mu_2, \rho) \in \mathcal{F}} \frac{1 + \theta\mu_1 + \theta\mu_2 - f(\mu_1, -\mu_2, -\rho)}{1 + \mu_1 + \mu_2 + \rho} \\ &= \min_{(\mu_1, \mu_2, \rho) \in \mathcal{F}} \frac{1 + \theta\mu_1 + \theta\mu_2 + f(\mu_1, \mu_2, \rho)}{1 + \mu_1 + \mu_2 + \rho} \\ &= \alpha\left(\theta; \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right). \end{aligned}$$

(c)

$$\begin{aligned} \alpha\left(\theta; \frac{3}{4}, -\frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right) &= \min_{(\mu_1, \mu_2, \rho) \in \mathcal{F}} \frac{3 - \theta\mu_1 + \theta\mu_2 + f(\mu_1, \mu_2, \rho)}{3 - \mu_1 + \mu_2 + \rho} \\ &= \min_{(\mu_1, \mu_2, \rho) \in \mathcal{F}} \frac{3 + \theta\mu_1 + \theta\mu_2 + f(-\mu_1, \mu_2, -\rho)}{3 + \mu_1 + \mu_2 - \rho} \\ &= \min_{(\mu_1, \mu_2, \rho) \in \mathcal{F}} \frac{3 + \theta\mu_1 + \theta\mu_2 - f(\mu_1, \mu_2, \rho)}{3 + \mu_1 + \mu_2 - \rho} \\ &= \alpha\left(\theta; \frac{3}{4}, \frac{1}{4}, \frac{1}{4}, -\frac{1}{4}\right). \end{aligned}$$

□

Finally, we estimate the quality of solution  $\{\hat{x}\}$ .

**Lemma 7.** ([1])

$$\Pr \left[ \left| \sum_{i \in V} \tilde{x}_i \right| \geq \frac{\varepsilon n}{10} \right] \leq \frac{\varepsilon}{10}.$$

Lemmas 1 and 7 imply the following lemma, whose proof is deferred to the journal version.

**Lemma 8.**

$$E \left[ w(\hat{S}) \right] \geq \left( 1 - \frac{\varepsilon}{2} \right) E \left[ w(\tilde{S}) \right].$$

Now we are able to present the approximation ratios of Algorithm 1 for the graph bisection problems in Table 3 by using a computer-assisted proof.

**Table 3.** Approximation ratios

Problem	Ratio
MAX- $\frac{n}{2}$ -CUT	0.8732 ( $\theta = 0.87$ )
MAX- $\frac{n}{2}$ -UNCUT	0.8732 ( $\theta = 0.87$ )
MAX- $\frac{n}{2}$ -DENSE-SUBGRAPH	0.8731 ( $\theta = 0.87$ )
MAX- $\frac{n}{2}$ -VERTEX-COVER	0.9401 ( $\theta = 0.94$ )
MAX- $\frac{n}{2}$ -DIRECTED-CUT	0.8731 ( $\theta = 0.87$ )
MAX- $\frac{n}{2}$ -DIRECTED-UNCUT	0.9401 ( $\theta = 0.94$ )

**Table 4.** Approximation ratios for the  $k$ -cut problems where the first two ratios 0.8582 are due to Raghavendra and Tan [8]

Problem	Ratio
MAX- $k$ -CUT	0.8582 ( $\theta = 1$ )
MAX- $k$ -UNCUT	0.8582 ( $\theta = 1$ )
MAX- $k$ -DENSE-SUBGRAPH	0.1079 ( $\theta = 1$ )
MAX- $k$ -VERTEX-COVER	0.9291 ( $\theta = 1$ )
MAX- $k$ -DIRECTED-CUT	0.1079 ( $\theta = 1$ )
MAX- $k$ -DIRECTED-UNCUT	0.9291 ( $\theta = 1$ )

Note that the first two ratios in Table 3 are not as good as the corresponding ratios in Table 1. The latter were obtained by a slightly involved rounding technique, namely, the pairing vertices selection algorithm in [1] for MAX- $\frac{n}{2}$ -CUT, and hence implying the same approximation ratio for MAX- $\frac{n}{2}$ -UNCUT via Lemma 6.

## 5 Discussions

In this paper, we present a unified SDPH rounding approximation algorithm for six types of maximum graph bisection problems. Analogous algorithm and analysis can be extended to the more general  $k$ -CUT problems (we assume that  $k/n$  is a positive constant), whose approximation ratios are illustrated in Table 4.

**Acknowledgments.** The authors would like to thank the anonymous referees for the insightful and helpful comments. The second author's research is supported by National Science and Engineering Research Council of Canada (NSERC) grants 283106. The third author's research is supported by NSF of China (No. 11071268) and Scientific Research Common Program of Beijing Municipal Commission of Education (No. KM201210005033).

## References

1. Austrin, P., Benabbas, S., Georgiou, K.: Better balance by being biased: a 0.8776-approximation for max bisection. In: Proceedings of SODA, pp. 277–294 (2013), Full version available as arXiv eprint 1205.0458v2
2. Feige, U., Langberg, M.: The RPR<sup>2</sup> rounding technique for semidefinite programs. *Journal of Algorithms* 60, 1–23 (2006)
3. Frieze, A.M., Jerrum, M.: Improved approximation algorithms for MAX  $k$ -CUT and MAX BISECTION. *Algorithmica* 18, 67–81 (1997)
4. Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM* 42, 1115–1145 (1995)
5. Halperin, E., Zwick, U.: A unified framework for obtaining improved approximation algorithms for maximum graph bisection problems. *Random Structures & Algorithms* 20, 382–402 (2002)
6. Han, Q., Ye, Y., Zhang, J.: An improved rounding method and semidefinite programming relaxation for graph partition. *Mathematical Programming, Series B* 92, 509–535 (2002)
7. Lasserre, J.B.: An explicit equivalent positive semidefinite program for nonlinear 0-1 programs. *SIAM Journal on Optimization* 12, 756–769 (2002)
8. Raghavendra, P., Tan, N.: Approximating CSPs with global cardinality constraints using SDP hierarchies. In: Proceedings of SODA, pp. 373–387 (2012), Full version available as arXiv eprint 1110.1064v1
9. Xu, D., Han, J., Huang, Z., Zhang, L.: Improved approximation algorithms for MAX  $n/2$ -DIRECTED-BISECTION and MAX  $n/2$ -DENSE-SUBGRAPH. *Journal of Global Optimization* 27, 399–410 (2003)
10. Ye, Y.: A .699-approximation algorithm for Max-Bisection. *Mathematical Programming* 90, 101–111 (2001)

# Improved Local Search for Universal Facility Location<sup>\*</sup>

Eric Angel, Nguyen Kim Thang, and Damien Regnault

IBISC, University of Evry Val d'Essonne, France  
`{angel,thang,regnault}@ibisc.univ-evry.fr`

**Abstract.** We consider the universal facility location problem in which the goal is to assign clients to facilities in order to minimize the sum of connection and facility costs. The connection cost is proportional to the distance each client has to travel to its assigned facility, whereas the cost of a facility is a non-decreasing function depending on the number of clients assigned to the facility. This model generalizes several variants of facility location problems. We present a  $(5.83 + \epsilon)$  approximation algorithm for this problem based on local search technique.

## 1 Introduction

The class of facility location problems is fundamental in operations research and is subject of extensive study. In the classical model, facilities are opened to satisfy client demands and the opening cost of a facility is fixed. However, this model is not fully appropriate in the contexts where the cost of a facility to serve clients (the delay) crucially depends on its allocated capacity. This phenomenon widely occurs in practical situations. The following model of Universal Facility Location captures this phenomenon and is also a generalization of several variants of facility location problems.

*Universal Facility Location.* Let  $\mathcal{C}$  be the set of (*clients*) and  $\mathcal{F}$  be a set of *facilities* where  $n = |\mathcal{C}|$  and  $m = |\mathcal{F}|$ . Each facility  $i$  is characterized by a non-decreasing *cost function*  $f_i : \mathbb{N} \rightarrow \mathbb{R}^+$  where  $f_i(0) = 0$ . Consider the complete bipartite graph  $G = (\mathcal{C} \cup \mathcal{F}, E)$  where the distances  $d(i, j)$  associated to facility  $i$  and client  $j$  follow the triangle inequality. We denote also by  $d(i, i')$  the length of a shortest path between two facilities  $i$  and  $i'$  in the graph.

The goal is to assign clients to the facilities and install capacities at every facility in order to serve clients. Given a *solution*  $S = (u, x)$ , where  $x$  is the *assignment* ( $x_{ij} = 1$  if client  $j$  is served by facility  $i$ ; and 0 otherwise) and  $u$  is the *allocation* ( $u_i \in \mathbb{N}$  denotes the capacity allocated at facility  $i$ , which equals the number of clients assigned to facility  $i$ ), the *connection cost* is defined as  $C_s(S) = \sum_{i \in \mathcal{F}, j \in \mathcal{C}} d(i, j)x_{ij}$  and the *facility cost* is  $C_f(S) = \sum_{i \in \mathcal{F}} f_i(u_i)$ . The objective is to find a feasible solution minimizing the *total cost*  $C(S)$  defined as  $C_s(S) + C_f(S)$ .

---

\* This work has been supported by ANR project TODO (09-EMER-010) and GdR RO.

*Related Works.* The model of Universal Facility Location captures several variants of Facility Location. Many interesting algorithms with deep, powerful techniques have been designed for the latter. Among others, the technique of local search is extensively studied. Arya et al. [1] introduced the local search technique to study Facility Location and  $k$ -median problem. From that, local search plays an important role to study variants of Facility Location.

A particular interesting variant is the Capacitated Facility Location Problem, where  $f_i(\cdot)$  is constant if the assigned amount to facility  $i$  is smaller than a given capacity; and is infinity otherwise. The first constant approximation ratio for this problem is 8.53 due to Pál and Tardos [7]. Then the approximation ratio for Capacitated Facility Location Problem was improved to  $(5.83 + \epsilon)$  by Zhang et al. [9] and recently to  $(5 + \epsilon)$  by Bansal et al. [2].

Mahdian and Pál [6] introduced the model of Universal Facility Location and gave a  $(7.88 + \epsilon)$ -approximation algorithm. Garg et al. [3] proposed extended operations and a schema of analysis to prove a  $(5.83 + \epsilon)$ -approximation ratio. However, one of their operations is unlikely to be polynomially computable (N. Garg, personal communication, 2012). Subsequently, Vygen [8] improved the approximation ratio to  $(6.702 + \epsilon)$ . All those algorithms are based on local search approach and the successive improvements are done by extending and generalizing the previous operations together with more subtle analyses.

Besides, Hajiaghayi et al. [4] considered Universal Facility Location with concave cost function and designed a 1.861-approximation algorithm. Recently, Li and Khuller [5] have proved a  $(\ln n + 1)$ -approximation for the Universal Facility Location in non-metric space.

*Contributions.* We present a  $(5.83 + \epsilon)$ -approximation algorithm also based on local search. The contribution of the paper is a simple, polynomially computable operation called **Open-close**. With this operation together with other operations, we manage to show the improved performance on the approximation of the Universal Facility Location. Note that the analysis follows closely the ones in [9,3] with **Open-close** as the main operation.

## 2 Algorithm and Analysis

### 2.1 Operations

In this section we describe the set of operations that will be used in the algorithm.

- **Add**( $s, \delta$ ): increase the capacity of facility  $s$  by  $\delta$ , and find the minimum cost assignment of demands to facilities, given their allocated capacities.
- **Open**( $s, \delta$ ): increase the capacity of  $s$  by sending  $\delta$  units of flow from one or several facilities  $i_1, i_2, \dots$  to  $s$  via the shortest paths between  $i_1, i_2, \dots$  and  $s$  (and decrease the capacity of  $i_1, i_2, \dots$ ).
- **Close**( $s, \delta$ ): Inversely, decrease the capacity of  $s$  by sending  $\delta$  units of flow from  $s$  to one or several facilities  $i_1, i_2, \dots$  via shortest path between  $s$  and  $i_1, i_2, \dots$  (and increase the capacity of  $i_1, i_2, \dots$ ).

- **Open-close**( $s, t, \delta_s, \delta_t$ ): increase the capacity of  $s$  by  $\delta_s$  and decrease the capacity of  $t$  by  $\delta_t$ . This operation consists in orienting some amount from  $t$  to  $s$  and then routing some amounts from one or several facilities  $i_1, i_2, \dots$  to  $s$  and from  $t$  to one or several facilities  $i'_1, i'_2, \dots$ . The transfers are carried out via shortest paths between facilities.

In the following for each operation, given its input, we show how to compute the minimum cost (of the operation on the input) in polynomial time. The min-cost of operation **Add**( $s, \delta$ ) has been shown to be efficiently computable [6,8]. Note that the operations **Open**( $s, \delta$ ) and **Close**( $s, \delta$ ) are particular cases of the operation **Open-close**( $s, t, \delta_s, \delta_t$ ), hence it is sufficient to prove that the min-cost of the latter could be computed in polynomial time.

**Lemma 1.** *Let  $S = (u, x)$  be a solution and  $s, t$  be two facilities and  $0 \leq \delta_s, \delta_t \leq n$ . Then, the minimum cost of the operation **Open-close**( $s, t, \delta_s, \delta_t$ ) can be computed in polynomial time with respect to  $n, m$ .*

*Proof.* Observe that if in operation **Open-close**( $s, t, \delta_s, \delta_t$ ), some amount is sent from  $t$  to some facility  $i$  and later is reoriented to  $s$  then we can modify the transfer in such a way that the amount is routed directly from  $t$  to  $s$ . Since the distance  $d$  follows the triangle inequality, the modification results in a solution at least as good as the previous one. Hence, in the sequel we assume that for any facility, either it receives some flow or it sends out some flow.

We compute the minimum cost of **Open-close**( $s, t, \delta_s, \delta_t$ ). Name  $1, \dots, m-2$  the facilities of  $\mathcal{F} \setminus \{s, t\}$ . Let  $0 \leq \delta \leq \min\{\delta_s, \delta_t\}$  be the amount of flow directly sent from  $t$  to  $s$ . We need to route  $\delta_s - \delta$  flow units to  $s$  and  $\delta_t - \delta$  units out of  $t$ . Let  $g(i, a, b)$  be the minimum cost of having already sent  $a$  flow units to  $s$  and having already sent  $b$  flow units out of  $t$  after considering the facilities  $1, 2, \dots, i$ . We have:

$$\begin{aligned} \text{Open-close}(s, t, \delta_s, \delta_t) &= \min_{0 \leq \delta \leq \min\{\delta_s, \delta_t\}} g(n-2, \delta_s - \delta, \delta_t - \delta) + \delta \cdot d(s, t) + \\ &\quad \left( f_s(u_s + \delta_s) - f_s(u_s) + f_t(u_t - \delta_t) - f_t(u_t) \right) \end{aligned}$$

Now we compute  $g(i, a, b)$  for  $1 \leq i \leq m-2$ ,  $0 \leq a \leq \delta_s - \delta$ ,  $0 \leq b \leq \delta_t - \delta$  by dynamic programming. At facility  $i$ , either  $i$  will transfer some amount to facility  $s$  or  $i$  will receive some amount from  $t$ . So we derive the recursive formula

$$g(i, a, b) = \min \left\{ \begin{array}{l} \min_{0 \leq w \leq a} g(i-1, a-w, b) + \left[ w \cdot d(i, s) + f_i(u_i - w) - f_i(u_i) \right], \\ \min_{0 \leq w \leq b} g(i-1, a, b-w) + \left[ w \cdot d(i, t) + f_i(u_i + w) - f_i(u_i) \right] \end{array} \right\}$$

for  $2 \leq i \leq n-2$ ,  $0 \leq a \leq \delta_s - \delta$ ,  $0 \leq b \leq \delta_t - \delta$  and

$$g(1, a, b) = \begin{cases} a \cdot d(1, s) + f_1(u_1 - a) - f_1(u_1) & \text{if } b = 0 \\ b \cdot d(1, t) + f_1(u_1 + b) - f_1(u_1) & \text{if } a = 0 \\ \infty & \text{if } a \neq 0, b \neq 0. \end{cases}$$

where the last case indicates that a facility can either receive or send out some amount but not both. As  $\delta_s$  and  $\delta_t$  are bounded by  $n$ , we can compute  $\text{Open-close}(s, t, \delta_s, \delta_t)$  in  $O(mn^4)$ .  $\square$

## 2.2 The Local Search Algorithm

Fix  $\epsilon > 0$  be a small constant. Let  $S$  be an arbitrary feasible solution. As long as there still exists some operation  $\text{Add}(s, \delta)$  for  $0 \leq \delta \leq \alpha$  or  $\text{Open-close}(s, t, \delta_s, \delta_t)$  for  $0 \leq \delta_s, \delta_t \leq \alpha$  (using Lemma 1) such that after the operation the cost is reduced by at least  $\epsilon C(S)$ , improve  $S$  by the operation. Otherwise, return  $S$ .

By the results of the previous section, at each step we can verify in polynomial time whether there is some improvement due to the operations. Moreover, the algorithm halts after at most  $\frac{1}{\epsilon} \log \frac{C(S)}{C(S^*)}$  iterations where  $S^*$  is a global optimum. Hence, the running time of the algorithm is polynomial in the size of the input.

## 2.3 The Analysis

Note that the solution returned by the algorithm is not a local optimum (according to the operation given in previous section), but is an approximate one. However, the cost of the latter is only  $(1 + 3\epsilon)$  factor worse than the bound of a local optimum. Hence, by standard argument in local search, it is sufficient to prove the bound  $r$  of a local optimum (with respect to the operations described in the previous section) to a global optimum. Consequently, the approximation ratio is  $r(1 + \epsilon')$  where  $\epsilon' = 3\epsilon$ .

Let  $S = (u, x)$  and  $S^* = (u^*, x^*)$  be a local optimum solution and a global optimum, respectively. With respect to the  $\text{Add}$  operation, the connection cost has been bounded by the following lemma.

**Lemma 2 ([6,8]).**  $C_s(S) \leq C_s(S^*) + C_f(S^*)$ .

The remaining of the paper is devoted to bound  $C_f(S)$  in function of  $C_s(S^*)$  and  $C_f(S^*)$  by the following strategy.

*Strategy.* Define  $\mathcal{F}^+ := \{i \in F : u_i > u_i^*\}$  and  $\mathcal{F}^- := \{i \in F : u_i < u_i^*\}$ . The idea of the proof is to transfer some capacity amounts from facilities in  $\mathcal{F}^+$  to facilities in  $\mathcal{F}^-$  based on the operations defined in the previous section while maintaining the following properties.

- For each facility  $i \in \mathcal{F}^+$ , move once the exact amount of  $(u_i - u_i^*)$  units from  $i$  to some facilities in  $\mathcal{F}^-$ . We say that facility  $i$  is *closed*.
- For each facility  $i \in \mathcal{F}^-$ , the amount that  $i$  receives each time is at most  $u_i^* - u_i$ . Each time  $i$  receives some capacity amount, we say that facility  $i$  is *opened*.
- The *transportation cost* of the transfer — the cost to route capacity amounts between facilities where each unit travelling from facility  $i$  to  $i'$  along a path incurs a cost as the total length of that path — is small.

Note that a transfer is not a sequence of successive operations but is a “union” of different operations. Ideally, in the second property each facility in  $\mathcal{F}^-$  is opened once. However, the operations fulfilling this purpose may not be computed in polynomial time. Let  $r$  be the maximum number of times a facility in  $\mathcal{F}^-$  is open in such a transfer. Suppose that there exists a transfer with the desired properties. We show how the strategy leads to useful bounds of the facility cost. Let  $C_t$  be the transportation cost of the transfer. As  $S$  is local optimum, any operation with respect to the solution  $S$  must have non-negative cost. Denote  $\delta_{i,r'}$  be the amount transferred to facility  $i \in \mathcal{F}^-$  at its  $r'$ -th opening. Combining all inequalities corresponding to operations in the transfer, we have

$$\sum_{i \in \mathcal{F}^+} (f_i(u_i^*) - f_i(u_i)) + \sum_{r'=1}^r \sum_{i \in \mathcal{F}^-} (f_i(u_i + \delta_{i,r'}) - f_i(u_i)) + C_t \geq 0. \quad (1)$$

Therefore,

$$\sum_{i \in \mathcal{F}^+} (f_i(u_i^*) - f_i(u_i)) + r \cdot \sum_{i \in \mathcal{F}^-} (f_i(u_i^*) - f_i(u_i)) + C_t \geq 0. \quad (2)$$

since for  $i \in \mathcal{F}^-$ ,  $f_i(u_i^*) \geq f_i(u_i + \delta_{i,r'})$ , which is due to  $\delta_{i,r'} \leq u_i^* - u_i$  by the second property, thus the lefthand side of (1) is upper-bounded by that of (2). Summing both sides of (2) by  $\sum_{i:i \notin \mathcal{F}^- \cup \mathcal{F}^+} f_i(u_i)$  and rearranging the terms, we get  $C_f(S) \leq rC_f(S^*) + C_t$ . Whenever  $C_t$  is small, we can derive a bound on the facility cost  $C_f(S)$ . We say that a transfer is *feasible* if it satisfies the first two properties. We will look for feasible transfers with small transportation cost and  $r$  as small as possible.

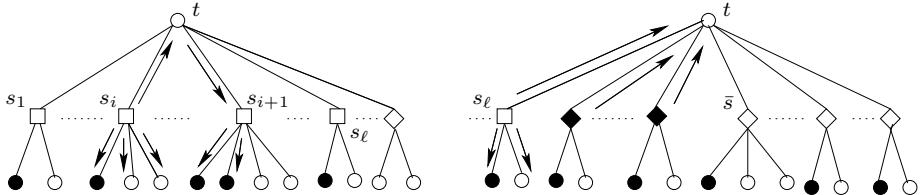
Consider a transportation problem: finding a min cost flow such that each facility  $i \in \mathcal{F}^+$  sends out  $u_i - u_i^*$  units of flow and each facility  $i \in \mathcal{F}^-$  receives  $u_i^* - u_i$  units. The cost of shipping one flow unit between  $i$  and  $i'$  equals  $d(i, i')$ . Mahdian and Pál [6] proved that the minimum cost flow was at most  $C_s(S) + C_s(S^*)$ . Moreover, the support graph of the min cost transportation forms a forest with edges going between  $\mathcal{F}^+$  and  $\mathcal{F}^-$ . The transfers that we will define later are carried out based on this forest.

Root each tree in the forest at some fixed facility in  $\mathcal{F}^-$ . For each vertex  $v$ , denote  $K(v)$  the set of its children. For each vertex  $t \in \mathcal{F}^-$ , let  $T_t$  be the subtree of depth *exactly* 2 rooted at  $t$  containing all its children and grand children. (We can add some dummy vertices where the in-flow and out-flow are 0 such that every tree  $T_t$  has depth 2.) Let  $y$  be the optimal flow of the transportation problem where  $y(s, t)$  the flow between  $s \in \mathcal{F}^+$  and  $t \in \mathcal{F}^-$ . We denote  $y(s, V^-) = \sum_{t \in V^-} y(s, t)$  and  $y(V^+, t) = \sum_{s \in V^+} y(s, t)$  for  $V^- \subset \mathcal{F}^-$  and  $V^+ \subset \mathcal{F}^+$ , respectively. For special cases where  $V^- = \mathcal{F}^-$  and  $V^+ = \mathcal{F}^+$ , we simply denote  $y(\cdot, t)$  as  $y(\mathcal{F}^+, t)$  and  $y(s, \cdot)$  as  $y(s, \mathcal{F}^-)$  the total flows received at  $t$  and the total flow sent from  $s$ , respectively. Hence, for  $s \in \mathcal{F}^+$  and  $t \in \mathcal{F}^-$   $y(\cdot, t) = u_t^* - u_t$  and  $y(s, \cdot) = u_s - u_s^*$ .

In the remaining, we will give a feasible transfer which closes each facility in  $\mathcal{F}^+$  once, opens each facility in  $\mathcal{F}^+$  at most three times and the transportation

cost of the transfer is also bounded by twice that of the optimal flow  $y$ . The transfer scheme follows the same scheme in [3] with **Open-close** as the main operation. For completeness, we present the schema in the following.

Consider a facility  $t$  of  $\mathcal{F}^-$  and the subtree  $T_t$ . Note that  $K(t) = T_t \cap \mathcal{F}^+$ . We will classify the facilities into groups indicating where the main part of the sent (received) flow amount goes to (comes from). We say that a facility  $t \in \mathcal{F}^-$  is *strong* if  $y(\cdot, t) \geq 2y(K(t), t)$ ; and *weak* otherwise. Intuitively,  $t$  is strong means that the main part of flow that  $t$  receives comes from its parent. Similarly, a facility  $s \in \mathcal{F}^+$  is *dominant* if  $y(s, t) \geq y(s, K(s))$ ; and *non-dominant* otherwise. Again, intuitively a dominant facility routes out the main part of its out-flow to its parent. In the tree  $T_t$ , let  $\text{Dom}(t)$  and  $\text{NDom}(t)$  be the sets of dominant and non-dominant facilities of  $K(t)$ , respectively. For each facility  $s \in K(t)$ , let  $S(s)$  and  $W(s)$  be the set of strong and weak facilities in  $K(s)$ , respectively.



**Fig. 1.** In the figures, the squares and diamonds represent non-dominant and dominant facilities in  $\mathcal{F}^+ \cap T_t$ , respectively. Besides, the circles and black circles represent weak and strong facilities in  $\mathcal{F}^- \cap T_t$ , respectively. The figure in the left illustrates operation  $\text{Close}(s_i, y(s_i, \cdot))$  in Step 1. The figure in the right illustrates operation  $\text{Open-close}(t, s_\ell, y(s_\ell, t) + \sum_{s \in \text{Dom}_2} y(s, \cdot), y(s_\ell, \cdot))$  in Step 2c where black diamonds stands for facilities in  $\text{Dom}_2$ .

Consider the transfer by performing the following operations. (An illustration is shown in Figure 1.)

1. For each facility  $s \in \text{NDom}(t)$ , define  $\text{Rem}(s) := \max\{y(s, t) - y(s, W(s)), 0\}$ . Order facilities in  $\text{NDom}(t)$  as  $\{s_1, \dots, s_\ell\}$  such that  $\text{Rem}(s_1) \leq \dots \leq \text{Rem}(s_\ell)$ . For  $i \in \{1, \dots, \ell - 1\}$ , consider  $\text{Close}(s_i, y(s_i, \cdot))$ : decrease the capacity at  $s_i$  by moving out  $2y(s_i, t')$  units to every facility  $t' \in W(s_i)$ ,  $y(s_i, t')$  units to every facility  $t' \in S(s_i)$  and  $\text{Rem}(s_i)$  units to facilities in  $S(s_{i+1})$ . The latter must be distributed in such a way that each facility in  $t' \in S(s_{i+1})$  receives at most  $y(s_{i+1}, t')$  units. That can always be done since  $\text{Rem}(s_i) \leq \text{Rem}(s_{i+1}) \leq y(s_{i+1}, K(s_{i+1})) - y(s_{i+1}, W(s_{i+1})) = y(s_{i+1}, S(s_{i+1}))$  where the second inequality is because  $s_{i+1}$  is non-dominant.
2. We adopt different procedures depending on different cases.
  - a. **If  $t$  is strong.**  
Consider  $\text{Open-close}(t, s_\ell, y(s_\ell, t) + \sum_{s \in \text{Dom}(t)} y(s, \cdot), y(s_\ell, \cdot))$ : facility  $t$  receives  $y(s, \cdot)$  units from each facility  $s \in \text{Dom}(t)$  in addition with  $y(s_\ell, t)$  units from  $s_\ell$ ; send out  $y(s_\ell, t')$  units from  $s_\ell$  to every facility  $t' \in K(s_\ell) \cup t$ .

- b. If  $t$  is weak and there is a facility  $h \in \text{Dom}(t)$  such that  $y(h, t) \geq y(\cdot, t)/2$ .

- Consider  $\text{Close}(h, y(h, \cdot))$ : decrease the capacity at  $h$  by sending out  $y(h, t')$  units to every facility  $t' \in K(h) \cup t$ .
- Consider  $\text{Open-close}(t, s_\ell, y(s_\ell, t) + \sum_{s \in \text{Dom}(t), s \neq h} y(s, \cdot), y(s_\ell, \cdot))$ : facility  $t$  receives  $y(s, \cdot)$  units from each facility  $s \in \text{Dom}(t) \setminus \{h\}$  in addition with  $y(s_\ell, t)$  units from  $s_\ell$ ; send out  $y(s_\ell, t')$  units from  $s_\ell$  to every facility  $t' \in K(s_\ell) \cup t$ .

- c. If  $t$  is weak and there is no facility  $h \in \text{Dom}(t)$  such that  $y(h, t) \geq y(\cdot, t)/2$ .

In this case, Zhang et al. [9] has proved that there exist a facility  $\bar{s} \in \text{Dom}(t)$  such that the set  $\text{Dom}(t) \setminus \{\bar{s}\}$  can be partitioned into  $\text{Dom}_1$  and  $\text{Dom}_2$  satisfying  $y(\bar{s}, t) + \sum_{s \in \text{Dom}_1} y(s, \cdot) \leq y(\cdot, t)$  and  $y(s_\ell, t) + \sum_{s \in \text{Dom}_2} y(s, \cdot) \leq y(\cdot, t)$ .

- Consider  $\text{Open-close}(t, \bar{s}, y(\bar{s}, t) + \sum_{s \in \text{Dom}_1} y(s, \cdot), y(\bar{s}, \cdot))$ : facility  $t$  receives  $y(s, \cdot)$  units from each facility  $s \in \text{Dom}_1$  in addition with  $y(\bar{s}, t)$  units from  $\bar{s}$ ; send out  $y(\bar{s}, t')$  units from  $\bar{s}$  to every facility  $t' \in K(\bar{s}) \cup t$ .
- Similarly, consider  $\text{Open-close}(t, s_\ell, y(s_\ell, t) + \sum_{s \in \text{Dom}_2} y(s, \cdot), y(s_\ell, \cdot))$ : facility  $t$  receives  $y(s, \cdot)$  units from each facility  $s \in \text{Dom}_2$  in addition with  $y(s_\ell, t)$  units from  $s_\ell$ ; send out  $y(s_\ell, t')$  units from  $s_\ell$  to every facility  $t' \in K(s_\ell) \cup t$ .

**Lemma 3.** *The transfer is feasible.*

*Proof.* Fix a tree  $T_t$ . By the transfer procedure, each facility  $s \in T_t \cap \mathcal{F}^+$  sends out exactly  $y(s, \cdot)$  units. It remains to prove that every facility  $t' \in T_t \cap \mathcal{F}^-$  receives each time at most  $y(\cdot, t')$ .

Consider  $t' \in T_t \cap \mathcal{F}^- \setminus \{t\}$  and suppose that  $t' \in K(s)$  for some  $s \in T_t \cap \mathcal{F}^+$ . By the transfer on tree  $T_t$ , at any operation, if  $t'$  is weak then it receives at most  $2y(s, t')$ , which is bounded by  $y(\cdot, t')$ ; otherwise (if  $t'$  is strong) the received amount is at most  $y(s, t')$ . In any case,  $t'$  receives at most  $y(\cdot, t')$ .

Consider the root  $t$  of tree  $T_t$ . Note that facility  $t$  only receives flow in Step 2 of the procedure. If  $t$  is strong, the total amount sent to  $t$  is

$$y(s_\ell, t) + \sum_{s \in \text{Dom}(t)} y(s, \cdot) \leq y(s_\ell, t) + \sum_{s \in \text{Dom}(t)} 2y(s, t) \leq 2y(K(t), t) \leq y(\cdot, t)$$

where the first and the last inequalities follow by the definition of dominant facilities and the fact that  $t$  is strong, respectively.

If  $t$  is weak and there is facility  $h \in \text{Dom}(t)$  such that  $y(h, t) \geq y(\cdot, t)/2$ . Facility  $t$  receives  $y(h, t)$  units in the operation that closes facility  $h$ . In the operation that opens facility  $t$  and closes facility  $s_\ell$ , the amount sent to  $t$  is

$$\begin{aligned} y(s_\ell, t) + \sum_{s \in \text{Dom}(t), s \neq h} y(s, \cdot) &\leq y(s_\ell, t) + \sum_{s \in \text{Dom}(t), s \neq h} 2y(s, t) \\ &\leq 2y(K(t) \setminus \{h\}, t) \leq 2(y(\cdot, t) - y(h, t)) \leq y(\cdot, t). \end{aligned}$$

If  $t$  is weak and there is no facility  $h \in \text{Dom}(t)$  such that  $y(h, t) \geq y(\cdot, t)/2$ . The amounts that  $i$  receives are  $y(\bar{s}, t) + \sum_{s \in \text{Dom}_1} y(s, \cdot)$  and  $y(s_\ell, t) + \sum_{s \in \text{Dom}_2} y(s, \cdot)$  by the first and second operations of this case, respectively. As mentioned in the transfer procedure, those amounts are both bounded by  $y(\cdot, t)$ . Hence, in any case,  $t$  receives at most  $y(\cdot, t)$  units at each operation.  $\square$

**Lemma 4.** *In the transfer, each facility in  $\mathcal{F}^+$  is closed exactly once, each facility in  $\mathcal{F}^-$  is opened at most three times and the flow across every edge  $e = (s, t)$  in the forest (the support graph of the optimal solution in the transportation problem) is at most  $2y(s, t)$ .*

*Proof.* Clearly by the decomposition of the forest into subtrees, a facility in  $\mathcal{F}^+$  belongs to exactly one subtree and a facility in  $\mathcal{F}^-$  belongs to at most two subtrees  $T_t$ . Hence, by the previous lemma, every facility in  $\mathcal{F}^+$  is closed exactly once. Consider a facility  $t \in \mathcal{F}^-$ . Let  $\tilde{t}$  be the grand parent of  $t$ . If  $t$  is strong then it will be opened once by the transfer procedure on tree  $T_t$  and it is opened at most twice during the procedure on tree  $T_{\tilde{t}}$ . If  $t$  is weak then it will be opened at most twice by the procedure on tree  $T_t$  and once by the procedure on tree  $T_{\tilde{t}}$ . In any case,  $t$  is opened at most three time in the transfer.

Observe that edges in the subtrees  $T_t$  for all  $t$  are disjoint, so the flow across an edge  $e$  in the transfer is the one across the edge in the transfer restricted on the subtree containing  $e$ . Consider an edge  $e$  in a subtree  $T_t$  and the following cases.

*Case 1:  $e = (s, t)$  where  $s \in T_t \cap \mathcal{F}^+$ .* If  $s$  is dominant, then the total flow routed along  $(s, t)$  is at most  $y(s, \cdot)$ , which is bounded by  $2y(s, t)$ . If  $s$  is non-dominant, suppose that  $s = s_i$  for some  $1 \leq i \leq \ell$  where  $\text{NDom}(t) = \{s_1, \dots, s_\ell\}$  ordered according to the  $\text{Rem}$  functions. The total flow across  $(s_i, t)$  is due to: (1) the operation closing  $s_{i-1}$ ; and (2) the one closing  $s_i$  (especially for  $s_\ell$ , that is an operation opening  $t$  and closing  $s_\ell$ ). The first operation sends through  $(s_i, t)$  a flow  $\text{Rem}(s_{i-1}) \leq \text{Rem}(s_i) \leq y(s_i, t)$ ; the second operation routes along  $(s_i, t)$  a flow at most  $y(s_i, t)$ . Therefore, the total flow across  $e$  is at most  $2y(e)$ .

*Case 2:  $e = (s, t')$  where  $s \in T_t \cap \mathcal{F}^+$  and  $t' \in T_{t'} \cap \mathcal{F}^- \setminus \{t\}$ .* If  $s \in \text{Dom}(t)$  then by the transfer procedure the flow routed through  $(s, t')$  is at most  $y(s, t')$ . If  $s = s_i$  is non-dominant and  $t'$  is weak then the flow across  $(s, t')$  is either  $2y(s, t')$  in case  $s \neq s_\ell$  or  $y(s, t')$  in case  $s = s_\ell$ . If  $s = s_i$  is non-dominant and  $t'$  is strong then the flow sending from  $s_{i-1}$  to  $t'$  is at most  $y(s_i, t')$ . So together with the flow  $y(s_i, t')$  routing from  $s_i$ , the total flow across  $(s_i, t')$  is at most  $2y(s_i, t')$ .

In summary, the flow across every edge  $e$  in the forest is at most  $2y(e)$ . The lemma follows.  $\square$

**Theorem 1.** *It holds that  $C_f(S) \leq 4C_s(S^*) + 5C_f(S^*)$  and  $C_s(S) \leq C_s(S^*) + C_f(S^*)$ . Consequently, the local search algorithm is  $(3 + 2\sqrt{2} + \epsilon)$ -approximation ( $\approx (5.83 + \epsilon)$ -approx).*

*Proof.* By the previous lemma, the transportation cost of the transfer is at most  $2(C_s(S) + C_s(S^*))$  and a facility of  $\mathcal{F}^-$  is opened at most three times. Therefore,

$$C_f(S) \leq 3C_f(S^*) + 2(C_s(S) + C_s(S^*)) \leq 4C_s(S^*) + 5C_f(S^*)$$

since  $C_s(S) \leq C_s(S^*) + C_f(S^*)$ . Hence, the total cost of solution  $S$  is

$$C_s(S) + C_f(S) \leq C_s(S^*) + C_f(S^*) + 4C_s(S^*) + 5C_f(S^*) = 5C_s(S^*) + 6C_f(S^*)$$

which yields an approximation ratio  $(6 + \epsilon)$ .

By a standard scaling technique, we multiply the cost function of every facility by a factor  $\lambda$  (to be defined later) then apply the algorithm to this instance. Let  $C'_f(\cdot)$  and  $C_f(\cdot)$  be the facility cost function of the modified and original solution. Note that  $C'_f(\cdot) = \lambda C_f(\cdot)$ . Let  $S$  a local optimum (on the modified instance) and  $S^*$  be any feasible solution. The same analysis leads to the following inequalities:  $C'_f(S) \leq 4C_s(S^*) + 5C'_f(S^*)$  and  $C_s(S) \leq C_s(S^*) + C'_f(S^*)$ . Therefore, the original total cost

$$\begin{aligned} C_f(S) + C_s(S) &= C'_f(S)/\lambda + C_s(S) \leq \frac{1}{\lambda}(4C_s(S^*) + 5C'_f(S^*)) + C_s(S^*) + C'_f(S^*) \\ &= (5 + \lambda)C_f(S^*) + \left(\frac{4}{\lambda} + 1\right)C_s(S^*). \end{aligned}$$

Choosing  $\lambda = 2\sqrt{2} - 2$ , we get  $C(S) \leq (3 + 2\sqrt{2})C(S^*)$ . Therefore, the approximation ratio is  $(3 + 2\sqrt{2} + \epsilon) \approx (5.83 + \epsilon)$ .  $\square$

## References

1. Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., Pandit, V.: Local search heuristics for  $k$ -median and facility location problems. *SIAM J. Computing* 33(3), 544–562 (2004)
2. Bansal, M., Garg, N., Gupta, N.: A 5-approximation for capacitated facility location. In: Epstein, L., Ferragina, P. (eds.) *ESA 2012*. LNCS, vol. 7501, pp. 133–144. Springer, Heidelberg (2012)
3. Garg, N., Khandekar, R., Pandit, V.: Improved approximation for universal facility location. In: Proc. 16th Symposium on Discrete Algorithms, pp. 959–960 (2005)
4. Hajiaghayi, M.T., Mahdian, M., Mirrokni, V.S.: The facility location problem with general cost functions. *Networks* 42(1), 42–47 (2003)
5. Li, J., Khuller, S.: Generalized machine activation problems. In: Proc. of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 80–94 (2011)
6. Mahdian, M., Pál, M.: Universal facility location. In: Di Battista, G., Zwick, U. (eds.) *ESA 2003*. LNCS, vol. 2832, pp. 409–421. Springer, Heidelberg (2003)
7. Pál, M., Tardos, É., Wexler, T.: Facility location with nonuniform hard capacities. In: Proc. 42nd Annual Symposium on Foundations of Computer Science, pp. 329–338 (2001)
8. Vygen, J.: From stars to comets: Improved local search for universal facility location. *Operations Research Letters* 35(4), 427–433 (2007)
9. Zhang, J., Chen, B., Ye, Y.: A multiexchange local search algorithm for the capacitated facility location problem. *Mathematics of Operations Research* 30(2), 389–403 (2005)

# Improved Approximation Algorithms for Computing $k$ Disjoint Paths Subject to Two Constraints

Longkun Guo<sup>1,2,\*,\*\*</sup>, Hong Shen<sup>1,3</sup>, and Kewen Liao<sup>3</sup>

<sup>1</sup> School of Information Science and Technology, Sun Yat-Sen University, China

<sup>2</sup> College of Mathematics and Computer Science, Fuzhou University, China

<sup>3</sup> School of Computer Science, University of Adelaide, Australia

lkguo@fzsu.edu.cn

**Abstract.** For a given graph  $G$  with positive integral cost and delay on edges, distinct vertices  $s$  and  $t$ , cost bound  $C \in Z^+$  and delay bound  $D \in Z^+$ , the  $k$  bi-constraint path ( $k$ BCP) problem is to compute  $k$  disjoint  $st$ -paths subject to  $C$  and  $D$ . This problem is known NP-hard, even when  $k = 1$  [4]. This paper first gives a simple approximation algorithm with factor-(2, 2), i.e. the algorithm computes a solution with delay and cost bounded by  $2 * D$  and  $2 * C$  respectively. Later, a novel improved approximation algorithm with ratio  $(1 + \beta, \max\{2, 1 + \ln \frac{1}{\beta}\})$  is developed by constructing interesting auxiliary graphs and employing the cycle cancellation method. As a consequence, we can obtain a factor-(1.369, 2) approximation algorithm by setting  $1 + \ln \frac{1}{\beta} = 2$  and a factor-(1.567, 1.567) algorithm by setting  $1 + \beta = 1 + \ln \frac{1}{\beta}$ . Besides, by setting  $\beta = 0$ , an approximation algorithm with ratio  $(1, O(\ln n))$ , i.e. an algorithm with only a single factor ratio  $O(\ln n)$  on cost, can be immediately obtained. To the best of our knowledge, this is the first non-trivial approximation algorithm for the  $k$ BCP problem that strictly obeys the delay constraint.

**Keywords:**  $k$ -disjoint bi-constraint path, NP-hard, bifactor approximation algorithm, auxiliary graph, cycle cancellation.

## 1 Introduction

In real networks, there are many applications that require quality of service and some degree of robustness simultaneously. Typically, the quality of service (QoS) related problem requires routing between the source node and the destination node to satisfy several constraints simultaneously, such as bandwidth, delay, cost and energy consumption. Nevertheless, in networks, some time-critical applications also require routing to remain functioning while edge or vertex failure

---

\* This project was supported by the Natural Science Foundation of Fujian Province (2012J05115), Doctoral Fund of Ministry of Education of China for Young Scholars (20123514120013) and Fuzhou University Development Fund (2012-XQ-26).

\*\* Corresponding author.

occurs. A common solution is to compute  $k$  disjoint paths that satisfy the QoS constraints, and use one path as an *active* path whilst the other paths as *backup* paths. The routing traffic is carried on the active path, and switched to the disjoint backup paths while an edge or vertex failure occurs on the active path. However, for some time-critical applications even the time to discover failures of routing and restore data transmission in backup paths is too long for them. For such applications, packages are routed via  $k$  paths simultaneously, and the traffic is switched from failed paths to functioning paths if edge or vertex failures occur, such that routing can tolerate  $k - 1$  edge (vertex) failures. Therefore, given cost and delay as the QoS constraints, the *disjoint QoS Path problem* arises as below:

**Definition 1.** *For a graph  $G = (V, E)$  and a pair of distinct vertices  $s, t \in V$ , a cost function  $c : E \rightarrow Z^+$ , a delay function  $d : E \rightarrow Z^+$ , a cost bound  $C \in Z^+$  and a delay bound  $D \in Z^+$ , the  $k$ -disjoint QoS Paths problem is to compute  $k$  disjoint  $st$ -paths  $P_1, \dots, P_k$ , such that  $\sum_{i=1, \dots, k} c(P_i) \leq C$  and  $d(P_i) \leq D$  for every  $i = 1, \dots, k$ .*

This problem is NP-hard even when all edges of  $G$  are with cost 0 [8], which results in the difficulty to approximate the  $k$ -disjoint QoS Paths problem. An alternative method is to compute  $k$  disjoint with total cost bounded by  $C$  and delay bounded by  $D$  (equal to  $kD$  in Definition 1), and then route the packages via the paths according to their urgency priority, i.e., route urgent packages via paths of low delay whilst deferrable ones via paths of high delay of the  $k$  disjoint paths. Therefore, The disjoint bi-constraint path problem arises as in the following:

**Definition 2.** *(The  $k$  disjoint bi-constraint path problem,  $kBCP$ ) For a graph  $G = (V, E)$  with a pair of distinct vertices  $s, t \in V$ , a cost function  $c : E \rightarrow R^+$ , a delay function  $d : E \rightarrow R^+$ , a cost bound  $C \in Z^+$  and a delay bound  $D \in R^+$ , the  $k$ -disjoint bi-constraint path problem is to calculate  $k$  disjoint  $st$ -paths  $P_1, \dots, P_k$ , such that  $\sum_{i=1, \dots, k} c(P_i) \leq C$  and  $\sum_{i=1, \dots, k} d(P_i) \leq D$ .*

This paper will focus on bifactor approximation algorithms for the  $kBCP$  problem, which are introduced as below:

**Definition 3.** *An algorithm  $A$  is a bifactor  $(\alpha, \beta)$ -approximation for the  $kBCP$  problem, if and only if for every instance of  $kBCP$ ,  $A$  computes  $k$  disjoint  $st$ -paths of which the delay sum and the cost sum are bounded by  $\alpha * D$  and  $\beta * C$  respectively.*

Since a  $\beta$ -approximation with the single factor ratio on cost is identical to a bifactor  $(1, \beta)$ -approximation, we use them interchangeably in the text.

## 1.1 Related Work

This  $kBCP$  problem is NP-hard even when  $k = 1$  [4]. To the best of our knowledge, this paper is the first one that presents non-trivial approximation algo-

rithms for the  $k$ BCP problem formally. However, a number of papers have addressed problems closely related to  $k$ BCP, in particular the  $k$  restricted shortest path problem ( $k$ RSP), which is to calculate  $k$  disjoint  $st$ -paths of minimum cost-sum under the delay constraint  $\sum_{i=1,\dots,k} d(P_i) \leq D$ . An algorithm with bifactor approximation ratio  $(2, 2)$  has been developed in [6] for general  $k$ , while no approximation solution that strictly obeys the delay (or cost) constraint is known even when  $k = 2$ . For a positive real number  $r$ , bifactor ratio of  $(1 + \frac{1}{r}, r(1 + \frac{2(\log r+1)}{r})(1 + \epsilon))$  and  $(1 + \frac{1}{r}, r(1 + \frac{2(\log r+1)}{r}))$  have been achieved respectively in [10,3] for the case  $k = 2$  and under the assumption that the delay of each path in the optimal solution of  $k$ RSP is bounded by  $\frac{D}{k}$ .

Special cases of this problem have been studied. When the delay constraint is removed, this problem is reduced to the min-sum problem, which is to calculate  $k$  disjoint paths with the total cost minimized. This problem is known polynomially solvable [11]. Moreover, when  $k = 1$ , the problem reduces to the single bi-constraint path (BCP) problem, which is known as the basic QoS routing problem [4] and admits full polynomial time approximation scheme (FPTAS) [4,9]. Recently, the single BCP problem is still attracting considerable interests of the researchers. The strongest result known is a  $(1 + \epsilon)$ -approximation due to Xue et al [14].

Additionally, when the cost constraint is removed, the disjoint QoS problem reduces to the length bounded disjoint path problem of finding two disjoint paths with the length of each path constrained by a given bound. This problem is a variant of the min-Max problem of finding two disjoint paths with the length of the longer path minimized. Both of the two problems are known to be  $NP$ -complete [8], and with the best possible approximation ratio of 2 in digraphs [8], which can be achieved by applying the algorithm for the min-sum problem in [11,12]. Contrastingly, the min-min problem of finding two paths with the length of the shorter path minimized is NP-complete and doesn't admit  $K$  approximation for any  $K \geq 1$  [5,13,2]. The problem remains NP-complete and admits no polynomial time approximation scheme in planar digraphs [7].

## 1.2 Our Techniques and Results

The main result of this paper is a factor- $(1 + \beta, \max\{2, 1 + \ln \frac{1}{\beta}\})$  approximation algorithm for any  $0 < \beta \leq 1$  for the  $k$ BCP problem. The main idea of the algorithm is firstly to compute  $k$ -disjoint paths with delay-sum bounded by  $\alpha D$  and cost-sum bounded by  $(2 - \alpha)*C$ , where  $0 \leq \alpha \leq 2$  is a real number, and secondly to improve the computed  $k$  paths by novelly combining cycle cancellation [10] and cost-bounded auxiliary graph construction [14]. The key technique to prove the algorithm's approximation ratio is using definite integral to compute a close form for the sum of the cost increment during the improving phase.

As a consequence of the main result, we can obtain a factor- $(1.369, 2)$  approximation algorithm by setting  $1 + \ln \frac{1}{\beta} = 2$ , and a factor- $(1.567, 1.567)$  algorithm by setting  $1 + \beta = 1 + \ln \frac{1}{\beta}$  and slightly modifying our algorithm (to improve either cost or delay that is with worse ratio). Nevertheless, by slightly modifying

**Algorithm 1.** A basic approximation algorithm for the  $k$ -BCP problem

**Input:** A graph  $G = (V, E)$ , each edge  $e$  with cost  $c(e)$  and delay  $d(e)$ , a given cost constraint  $C \in Z^+$  and delay constraint  $D \in Z^+$ ;

**Output:**  $k$  disjoint paths  $P_1, P_2 \dots, P_k$ .

1. Set the new cost of edge  $e$  as  $b(e) = \frac{c(e)}{C} + \frac{d(e)}{D}$ ;
2. Compute the  $k$  disjoint paths  $P_1, P_2 \dots, P_k$  in  $G$  by using Suurballe and Tarjan's algorithm [11,12], such that  $\sum_{i=1}^k \sum_{e \in P_i} b(e)$  is minimized;
3. Return  $P_1, P_2 \dots, P_k$ .

our ratio proof, we show that an approximation algorithm with ratio  $(1, O(\ln n))$ , i.e. an algorithm with single factor ratio of  $O(\ln n)$  on cost, can be immediately obtained by setting  $\beta = 0$ . To the best of our knowledge, this is the first non-trivial approximation algorithm for the  $k$ BCP problem that strictly obeys the delay constraint.

We note that our algorithms are with pseudo-polynomial time complexity, since the auxiliary graph we construct is of size  $O(C * n)$ . However, by using the classic polynomial time approximation scheme design technique [4], i.e. for any small  $\epsilon > 0$  setting the cost of every edge to  $\left\lfloor \frac{c(e)}{\frac{C}{n}} \right\rfloor$  in  $G$  before the construction of auxiliary graph, we can immediately obtain a polynomial time algorithm with ratio  $((1 + \beta) * (1 + \epsilon), \max\{2, 1 + \ln \frac{1}{\beta}\} * (1 + \epsilon))$ . We shall omit the details due to the paper length limitation.

## 2 An Improved Approximation Algorithm for Computing $k$ Disjoint Bi-constraint Paths

This section will first present a simple approximation method for computing  $k$ -disjoint paths with delay-sum bounded by  $\alpha D$  and cost-sum bounded by  $(2 - \alpha) * C$ , where  $0 \leq \alpha \leq 2$  is a real number, and secondly improve the computed  $k$  paths by balancing the value of  $\alpha$  and  $2 - \alpha$ . Though the presented simple algorithm is with worse ratio than that of the algorithm for  $k = 2$  in [10], it suits the improving phase better.

### 2.1 A Basic Approximation Algorithm

Observing that the difficulty of computing  $k$ -disjoint bi-constraint paths mainly comes from the two given constraints, the key idea of our algorithm is to deal with one new constraint  $B$  instead of the two given constraints  $C$  and  $D$ . Our algorithm firstly assigns a new mixed cost  $b(e) = \frac{c(e)}{C} + \frac{d(e)}{D}$  to every edge in graph, and secondly computes  $k$  disjoint paths with the new cost sum bounded by  $B = \frac{C}{C} + \frac{D}{D} = 2$ . Note that the second step can be accomplished in polynomial time by employing the SPP algorithm due to Suurballe and Tarjan [11,12]. The detailed algorithm is as in Algorithm 1.

The time complexity and performance guarantee of Algorithm 1 is given by the following theorem:

**Theorem 4** *Algorithm 1 runs in  $O(km \log_{1+\frac{m}{n}} n)$  time, and computes  $k$ -disjoint paths with delay-sum bounded by  $\alpha D$  and cost-sum bounded by  $(2-\alpha)*C$ , where  $0 \leq \alpha \leq 2$  is a real number.*

*Proof.* The main part of Algorithm 1 takes  $O(km \log_{1+\frac{m}{n}} n)$  to compute  $k$ -disjoint paths by using Surrballe and Tarjan's algorithm [11,12], and other parts of the algorithm take trivial time. Hence the time complexity of the algorithm is  $O(km \log_{1+\frac{m}{n}} n)$ .

It remains to show the approximation ratio. To make the proof concise, we denote by  $OPT$  an optimal solution for the  $k$ -disjoint BCP paths problem, and  $SOL$  the solution of Algorithm 1. Obviously  $\sum_{e \in OPT} b(e) \leq 2$  holds. Then since the  $k$  disjoint paths is with minimum new cost, we have

$$\sum_{e \in SOL} b(e) \leq \sum_{e \in OPT} b(e) \leq 2. \quad (1)$$

Assume the delay-sum of the algorithm is  $\alpha$  times of  $d(OPT)$ , then following Algorithm 1  $0 \leq \alpha \leq 2$  holds. Therefore, we have  $\sum_{e \in SOL} b(e) = \sum_{i=1}^k \sum_{e \in P_i} b(e) = \alpha + \frac{c(SOL)}{c(OPT)}$ . From Inequality (1),  $\alpha + \frac{c(SOL)}{c(OPT)} \leq 2$  holds. That is,  $c(SOL) \leq (2-\alpha)c(OPT) \leq (2-\alpha)C$ . This completes the proof.

Note that  $\alpha$  differs for different instances, i.e. Algorithm 1 may return a solution with cost  $2 * c(OPT)$  and delay 0 for some instances, while a solution with cost 0 and delay  $2 * d(OPT)$  for other instances. Hence, the bifactor approximation ratio for Algorithm 1 is actually  $(2, 2)$ .

In real networks, the two given constraints may not be of equal importance, say, delay is far more important comparing to cost. In this case, applications require that the delay of the resulting solution is bounded by  $(1+\beta)D$ , where  $0 < \beta < 1$  is a positive real number. Apparently, we could get an algorithm similar to Algorithm 1 excepting setting the new cost as  $b(e) = \beta \frac{c(e)}{C} + \frac{d(e)}{D}$ . The ratio of the new algorithm is given as below:

**Corollary 5.** *By setting the new cost as  $b(e) = \beta \frac{c(e)}{C} + \frac{d(e)}{D}$  for a given real number  $0 < \beta < 1$ , Algorithm 1 returns  $k$  paths with delay-sum bounded by  $\alpha D$  and cost-sum bounded by  $\frac{1+\beta-\alpha}{\beta} * C$ , where  $0 \leq \alpha \leq 1 + \beta$  is a real number. Therefore the ratio of the algorithm is  $(1+\beta, 1 + \frac{1}{\beta})$ .*

The proof of Corollary 5 is omitted here, since it is very similar to the proof of Theorem 1. According to Corollary 5, our algorithm can bound the delay-sum of the  $k$ -disjoint path by  $(1+\beta)D$  for any  $0 < \beta < 1$ , by relaxing the cost constraint to  $(1 + \frac{1}{\beta}) * C$ . For example, if  $\beta = 0.01$ , then the bifactor approximation ratio of the algorithm is  $(1.01, 101)$ . Thus, the algorithm decrease the delay of the  $k$ -disjoint paths at a high price. In the next subsection, we shall develop an improved method that pays less to make delay-sum of the  $k$ -disjoint paths bounded by  $(1+\beta)D$ .

---

**Algorithm 2.** An improved algorithm based on cycle cancellation.

---

**Input:** A graph  $G = (V, E)$ , each edge  $e$  with cost  $c(e)$  and delay  $d(e)$ , a given cost constraint  $C \in Z^+$  and delay constraint  $D \in Z^+$ , disjoint QoS paths  $P_1, P_2 \dots, P_k$  computed by Algorithm 1;

**Output:** Improved disjoint QoS paths  $Q_1, Q_2 \dots, Q_k$ .

1. **If**  $\sum_{i=1}^k d(P_i) \leq (1 + \beta)D$  ;  
**then** return  $P_1, P_2 \dots, P_k$  as  $Q_1, Q_2 \dots, Q_k$  , terminate;
  2. Reverse direction of the edges of  $P_1, P_2 \dots, P_k$  in  $G$  , set their cost to a small positive real number  $0 < \epsilon < \frac{1}{mnD}$ , and negative their delay;
  3. Compute cycle  $O_j$  with  $c(O_j) \leq C$ ,  $d(O_j) < 0$  and  $\frac{d(O_j)}{c(O_j)}$  attaining minimum, by the method given in next section;  
/\* Following clause 2 of Proposition 6, if  $\sum_{i=1}^k d(P_i) \geq d(OPT)$  and  $\sum_{i=1}^k c(P_i) \geq 0$ , there always exist cycle  $O_j$  with  $c(O_j) \leq C$  and  $d(O_j) < 0$ . \*/
  4. Improve  $P_1, P_2 \dots, P_k$  by adding the edges of  $O_j$  and removing the pairs of parallel edges in opposite direction;
  5. Go to Step 1.
- 

## 2.2 The Improving Phase

To make the delay of the solution resulting from Algorithm 1 bounded by  $(1 + \beta)D$ , our improving phase is, basically a greedy method, using the so-called cycle cancellation to improve the disjoint paths in iterations until a solution with the best possible ratio  $(1 + \beta, \max\{2, 1 + \ln \frac{1}{\beta}\})$  is obtained. The cycle cancellation method is an approach of using cycles to change the edges of the disjoint paths, which first appears in [10] and is derived from the following proposition that can be immediately obtained from flow theory [1]:

**Proposition 6.** Let  $P_1, P_2 \dots, P_k$  and  $Q_1, Q_2 \dots, Q_k$  be two sets of  $k$  disjoint st-paths in  $G$ ,  $\overline{G}$  be  $G$  excepting that all edges of  $P_1, P_2 \dots, P_k$  are reversed, and  $O$  be a cycle in  $\overline{G}$ . Then

1. The edges of  $P_1, P_2 \dots, P_k$  and  $O$ , excepting the pairs of parallel edges with opposite direction, compose  $k$ -disjoint paths;
2. There exist a set of edge disjoint cycles  $O_1, \dots, O_h$  in  $\overline{G}$ , such that the edges of  $P_1, P_2 \dots, P_k$  and  $O_1, \dots, O_h$ , excepting the pairs of parallel edges with opposite direction, compose  $Q_1, Q_2 \dots, Q_k$ .

From the proposition above, it is obvious that there exists a set of cycles  $O_1, \dots, O_h$  that can improve  $k$  disjoint QoS paths  $P_1, P_2 \dots, P_k$  to an optimal solution. However, it is hard to identify all the cycles  $O_1, \dots, O_h$ , so we employ a greedy approach to compute a set of cycles to obtain an approximation approach. The improving phase is composed by iterations, each of which computes a cycle and then uses it to improve  $P_1, P_2 \dots, P_k$ . More precisely, to obtain a good ratio, the algorithm computes in iteration  $j$  a cycle  $O_j$  with  $\frac{d(O_j)}{c(O_j)}$  minimized among the cycles in  $\overline{G}$ . The layout of the algorithm is as given in Algorithm 2.

Following clause 1 of Proposition 6, Algorithm 2 will correctly return  $k$  disjoint paths. It remains to show the cost and delay of the  $k$  disjoint paths is constrained as below:

**Theorem 7** *The approximation ratio of Algorithm 2 is  $(1 + \beta, \max\{2, 1 + \ln \frac{1}{\beta}\})$ .*

*Proof.* For the case that  $\sum_{i=1}^k d(P_i) \leq (1 + \beta)D$  holds before the improving phase, the approximation ratio of Algorithm 2 is obviously  $(1 + \beta, 2)$ .

It remains to show the ratio of the algorithm is  $(1 + \beta, 1 + \ln \frac{1}{\beta})$  for the case that  $\sum_{i=1}^k d(P_i) > (1 + \beta)D$ . Assume that Algorithm 2 runs in  $h$  iterations, the key idea of the proof is to sum up the cost increment while using the cycle to improve the  $k$  disjoint paths in iterations, and show that the cost sum is bounded (by giving the cost sum a close form).

Note that in the case, we have  $\alpha D \geq \sum_{i=1}^k d(P_i) > (1 + \beta)D$ , so  $\alpha > 1 + \beta$  holds. Let  $\Delta D = d(OPT) - d(SOL) \geq (1 - \alpha)d(OPT)$  and  $\Delta C = c(OPT) - c(SOL) \leq (\alpha - 1)c(OPT)$ . Clearly,  $\Delta D < 0$  and  $\Delta C > 0$  hold. Let the cycle computed in the  $j$ th iteration be  $O_j$ , then since  $\frac{d(O_j)}{c(O_j)}$  attains minimum in Step 3 of Algorithm 2, we have  $\frac{d(O_j)}{c(O_j)} \leq \frac{\Delta D - \sum_{i=1}^{j-1} d(O_i)}{C}$ . That is,

$$c(O_j) \leq \frac{d(O_j)}{\Delta D - \sum_{i=1}^{j-1} d(O_i)} C.$$

By summing up  $c(O_j)$  in  $h - 1$  iterations (excluding the last iteration), we have:

$$\sum_{j=1}^{h-1} c(O_j) \leq C \sum_{j=1}^{h-1} \frac{d(O_j)}{\Delta D - \sum_{i=1}^{j-1} d(O_i)}.$$

Following the definition of Definite Integral, we have:

$$\sum_{j=1}^{h-1} \frac{d(O_j)}{\Delta D - \sum_{i=1}^{j-1} d(O_i)} = \sum_{j=1}^{h-1} \frac{1}{\Delta D - \sum_{i=1}^{j-1} d(O_i)} d(O_j) \leq \int_{\Delta D}^{\Delta D - \sum_{i=1}^{h-1} d(O_i)} \frac{1}{x} dx, \quad (2)$$

where the maximum is attained when  $d(O_j) = -1$  for every  $j$ .

Algorithm 2 terminates when  $d(SOL) + \sum_{i=1}^h d(O_i) \leq (1 + \beta)D$ , so in the  $h - 1$  iterations  $d(SOL) + \sum_{i=1}^{h-1} d(O_i) > (1 + \beta)D$  holds. That is  $d(SOL) - D + \sum_{i=1}^{h-1} d(O_i) > \beta D$ , and hence  $-\Delta D + \sum_{i=1}^{h-1} d(O_i) > \beta D > 0$  holds. So we obtain a close form for the cost sum of the  $h - 1$  iterations:

$$\int_{\Delta D}^{\Delta D - \sum_{i=1}^{h-1} d(O_i)} \frac{1}{x} dx = \int_{-\Delta D + \sum_{i=1}^{h-1} d(O_i))}^{-\Delta D} \frac{1}{x} dx \leq \int_{\beta D}^{-\Delta D} \frac{1}{x} dx = \ln \frac{|\Delta D|}{\beta D} = \ln \frac{\alpha - 1}{\beta}. \quad (3)$$

At last, the cost increment in the  $h$ th iteration is bounded by  $c(OPT)$ . So the final cost is  $c(SOL_2) \leq (2 - \alpha)C + C \ln \frac{\alpha-1}{\beta} + C = C(3 - \alpha + \ln \frac{\alpha-1}{\beta})$ , where  $SOL_2$  is the solution resulting from Algorithm 2.

Let  $f(\alpha) = 3 - \alpha + \ln \frac{\alpha-1}{\beta}$ . Remind that  $\alpha \leq 2$ , so  $f'(\alpha) = \frac{1}{\alpha-1} - 1 > 0$ ,  $f(\alpha)$  is monotonous increasing on  $\alpha$ , and attains maximum while  $\alpha = 2$ . So we have  $c(SOL_2) \leq (1 + \ln \frac{1}{\beta})c(OPT)$ .

Therefore, the cost of the output of Algorithm 2 is bounded by  $(1 + \ln \frac{1}{\beta})c(OPT)$ , and delay bounded by  $(1 + \beta)d(OPT)$ . This completes the proof.

From Theorem 7, by setting  $1 + \ln \frac{1}{\beta} = 2$ , we can immediately obtain an improved algorithm with best possible delay ratio under the same cost bound  $2C$ . That is:

**Corollary 8.** *By setting  $1 + \ln \frac{1}{\beta} = 2$ , we have  $\beta = \frac{1}{e}$ , and hence Algorithm 2 is now with a bifactor approximation ratio of  $(1 + \frac{1}{e}, 2) = (1.369, 2)$ .*

For those applications in which delay and cost are of equal importance, by setting  $1 + \ln \frac{1}{\beta} = 1 + \beta$  and slightly modifying Algorithm 2 to improve either cost or delay that is of worse ratio, we can obtain an improved algorithm with ratio as in the following corollary:

**Corollary 9.** *If  $1 + \ln \frac{1}{\beta} = 1 + \beta$ , Algorithm 2 is with a bifactor approximation ratio of  $(1.567, 1.567)$ .*

Now we consider the case that  $\beta = 0$ , i.e. the delay constraint is strictly satisfied. In this case, Inequality (3) in the proof of Theorem 7 will become  $\sum_{j=1}^{h-1} \frac{d(O_j)}{\Delta D - \sum_{i=1}^{j-1} d(O_i)} \leq \int_{|\beta D|=0}^{|\Delta D|} \frac{1}{x} dx = \ln |\Delta D| \leq \ln D$ . So we have:

**Corollary 10.** *When  $\beta = 0$ , Algorithm 2 is with a ratio of  $(1, O(\ln n))$ .*

From Corollary 10, we can see that the price of obeying one constraint strictly is very high, i.e. it requires extra  $O(\ln n)$  times of cost. However, this is the first algorithm with logarithmic factor approximation ratio for the  $k$ -BCP problem with strict delay constraint.

### 3 Computing Cycle $O_j$ with Minimum $\frac{d(O_j)}{c(O_j)}$

Let  $\overline{G} = (V, E)$  be  $G$ , excepting that the edges of  $P_1, P_2 \dots, P_k$  are with direction reversed, cost set to 0, and delay negated. This section will show how to compute a cycle  $O$  with cost bounded by  $C$  and  $\frac{d(O)}{c(O)}$  minimized in  $\overline{G}$ . The key idea is firstly to construct an auxiliary graphs  $H(v)$  for each  $v$  where every cycle is with cost at most  $C$ , secondly to compute the cycle  $O'$  with minimum  $\frac{d(O')}{c(O')}$  among all cycles in all  $H(v)$ s for each  $v \in \overline{G}$ , and thirdly to obtain cycle  $O$  with minimum  $\frac{d(O(v))}{c(O(v))}$  in  $\overline{G}$  according to  $O'$ .

**Algorithm 3.** Construction of auxiliary graph  $H$ .

**Input:** Graph  $\overline{G} = (V, E)$ , two distinct vertices  $s, t \in V$ , a cost  $c : e \rightarrow Z_0^+$  and a delay  $d : e \rightarrow Z_0^+$  on every edge  $e \in E$ , a cost constraint  $C$  and a delay constraint  $D$ ;  
**Output:** Auxiliary graph  $H(v)$ .

1. For every vertex  $v_l$  of  $V$ , add to  $H(v)$  vertices  $v_l^1, \dots, v_l^C$ ;
2. For every edge  $e = \langle v_j, v_l \rangle \in E$ , add to  $H(v)$  the edges  $\langle v_j^1, v_l^{c(e)+1} \rangle, \dots, \langle v_j^{C-c(e)}, v_l^C \rangle$ , each of which is with cost  $c(e)$  and delay  $d(e)$ ;  
*/\*Note that  $d(e)$  can be negative in  $\overline{G} = (V, E)$ .\*/*
3. For all  $i = 2, \dots, C$ , add to  $H(v)$  backward edge  $\langle v^i, v^1 \rangle$  with delay 0 and cost 0, where a backward edge is an edge  $\langle v^i, v^j \rangle$  where  $i > j$ .  
*/\* $H(v)$  contains backward edges, and hence cycles, only after adding the edges of Step 3.\*/*

### 3.1 Construction of Auxiliary Graph $H(v)$

The algorithm of constructing the auxiliary graph  $H(v)$  is inspired by the method of computing a single path subject to multiple constraints [14]. The full layout of the algorithm is as shown in Algorithm 3 (An example of such construction is as depicted in Figure 1).

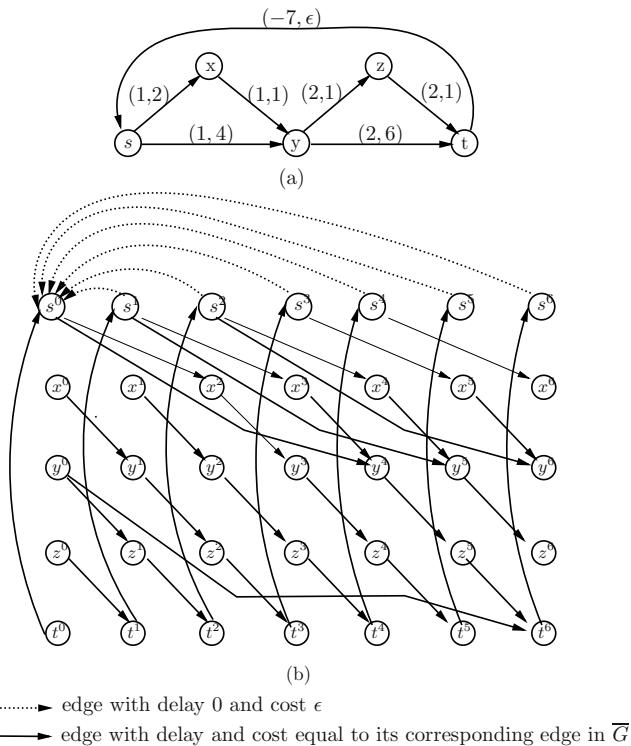
Following Algorithm 3, every backward edge in the constructed auxiliary graph  $H(v)$  must contain vertex  $v^1$ . Hence every cycle in  $H(v)$  contains at most one backward edge. On the other hand, following Algorithm 3 a cycle in  $H(v)$  contains at least one backward edge. Therefore, there exist exactly one backward edge in any cycle of  $H(v)$ . Because  $H(v) \setminus \{\langle v^2, v^1 \rangle, \dots, \langle v^C, v^1 \rangle\}$  is an acyclic graph where any path is with cost at most  $C$ , we have:

**Lemma 11.** *Any cycle in  $H(v)$  is with cost at most  $C$ .*

Let  $O(v)$  be a cycle in  $H(v)$ , then following the construction of  $H(v)$ ,  $O(v)$  apparently corresponds to a set of cycles in  $\overline{G}$ . Conversely, every cycle containing  $v$  in  $\overline{G}$  corresponds to a cycle in  $H(v)$ . Based on the observation, the following lemma gives the key idea of computing a cycle  $O$  of  $\overline{G}$  with  $\frac{d(O)}{c(O)}$  minimized and cost bounded by  $C$ :

**Lemma 12.** *Let  $O(v_i)$  be a cycle with minimum  $\frac{d(O(v_i))}{c(O(v_i))}$  in  $H(v_i)$ , and  $O(v)$  be the cycle with minimum  $\frac{d(O(v))}{c(O(v))}$  among the  $n$  cycles  $O(v_1), \dots, O(v_n)$ . Assume  $O$  is a cycle with minimum  $\frac{d(O)}{c(O)}$  in the set of cycles in  $\overline{G}$  that correspond to  $O(v)$ . Then for any cycle  $O'$  in  $\overline{G}$  with  $c(O') \leq C$ ,  $\frac{d(O)}{c(O)} \leq \frac{d(O')}{c(O')}$  holds.*

*Proof.* Suppose this lemma is not true, then there must exist in  $\overline{G}$  a cycle, say  $O'$ , such that  $\frac{d(O)}{c(O)} > \frac{d(O')}{c(O')}$  and  $c(O') \leq C$  hold. Then the cycle  $O'(v)$  in  $H(v)$  that corresponds to  $O'$  is also with  $\frac{d(O'(v))}{c(O'(v))} = \frac{d(O')}{c(O')} < \frac{d(O)}{c(O)} \leq \frac{d(O(v))}{c(O(v))}$ , contradicting with the minimality of  $O(v)$  in  $H(v)$ . This completes the proof.



**Fig. 1.** Construction of auxiliary graph  $H(v = s)$  with cost constraint  $C = 6$ : (a) graph  $\overline{G}$ ; (b) auxiliary graph  $H(v = s)$ . The cycle  $O = syts$  in  $\overline{G}$  is excluded in the auxiliary graph  $H(s)$  as shown in (b), keeping the cost of  $k$  disjoint paths constrained by  $C = 6$ .

### 3.2 Computing the Cycle $O$ with Minimum $\frac{d(O)}{c(O)}$

The main idea of the algorithm to compute a cycle  $O$  with  $\frac{d(O)}{c(O)}$  minimized in  $\overline{G}$  is to compute the cycle  $O'$  with minimum  $\frac{d(O')}{c(O')}$  among all cycles in all  $H(v)$ s for each  $v \in \overline{G}$ . Following Lemma 12, the cycle  $O$  in  $\overline{G}$  is the cycle with minimum  $\frac{d(O)}{c(O)}$  among the cycles in  $\overline{G}$  corresponding to all the computed  $O'$ 's. The detailed steps are as below:

1. For  $i = 1$  to  $n$ 
  - (a) Construct  $H(v_i)$  for  $v_i \in \overline{G}$  by Algorithm 3;
  - (b) Compute cycle  $O(v_i)$  with minimum  $\frac{d(O(v_i))}{c(O(v_i))}$  in  $H(v_i)$  by employing the minimum cost-to-time ratio cycle algorithm in [1];
  - (c) Select  $O(v)$  with minimum  $\frac{d(O(v))}{c(O(v))}$  from the  $n$  computed cycles  $O(v_1), \dots, O(v_n)$ ;
2. Select the cycle  $O$  with minimum  $\frac{d(O)}{c(O)}$  among the cycles in  $\overline{G}$  that correspond to  $O(v)$ .

Clearly, the cycle  $O$  attains minimum  $\frac{d(O)}{c(O)}$  in  $\overline{G}$ . Besides, following Lemma 11 we have  $c(O) \leq C$ . Therefore the cycle  $O$  is correctly the promised cycle. This completes the proof of the approximation ratio.

## 4 Conclusion

This paper gave a novel approximation algorithm with ratio  $(1 + \beta, \max\{2, 1 + \ln \frac{1}{\beta}\})$  for the  $k$ BCP problem based on improving a simple  $(\alpha, 2 - \alpha)$ -approximation algorithm by constructing interesting auxiliary graphs and employing the cycle cancellation method. By setting  $\beta = 0$ , an approximation algorithm with bifactor ratio  $(1, O(\ln n))$ , i.e. an  $O(\ln n)$ -approximation algorithm can be obtained immediately. To the best of our knowledge, it is the first non-trivial approximation algorithm for this problem that obeys the delay constraint strictly. We are now investigating whether any constant factor approximation algorithm exists for computing a solution that strictly obey the delay constraint.

## References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network flows: theory, algorithms, and applications (1993)
2. Bhatia, R., Kodialam, M., Lakshman, T.V.: Finding disjoint paths with related path costs. Journal of Combinatorial Optimization 12(1), 83–96 (2006)
3. Chao, P., Hong, S.: A new approximation algorithm for computing 2-restricted disjoint paths. IEICE Transactions on Information and Systems 90(2), 465–472 (2007)
4. Garey, M.R., Johnson, D.S.: Computers and intractability. Freeman, San Francisco (1979)

5. Guo, L., Shen, H.: On Finding Min-Min disjoint paths. accepted by Algorithmica
6. Guo, L., Shen, H.: Efficient approximation algorithms for computing k disjoint minimum cost paths with delay constraint. In: IEEE PDCAT, pp. 627–631. IEEE (2012)
7. Guo, L., Shen, H.: On the complexity of the edge-disjoint min-min problem in planar digraphs. *Theoretical Computer Science* 432, 58–63 (2012)
8. Li, C.L., McCormick, T.S., Simich-Levi, D.: The complexity of finding two disjoint paths with min-max objective function. *Discrete Applied Mathematics* 26(1), 105–115 (1989)
9. Lorenz, D.H., Raz, D.: A simple efficient approximation scheme for the restricted shortest path problem. *Operations Research Letters* 28(5), 213–219 (2001)
10. Orda, A., Sprintson, A.: Efficient algorithms for computing disjoint QoS paths. In: IEEE INFOCOM, vol. 1, pp. 727–738. Citeseer (2004)
11. Suurballe, J.W.: Disjoint paths in a network. *Networks* 4(2) (1974)
12. Suurballe, J.W., Tarjan, R.E.: A quick method for finding shortest pairs of disjoint paths. *Networks* 14(2) (1984)
13. Xu, D., Chen, Y., Xiong, Y., Qiao, C., He, X.: On the complexity of and algorithms for finding the shortest path with a disjoint counterpart. *IEEE/ACM Transactions on Networking* 14(1), 147–158 (2006)
14. Xue, G., Zhang, W., Tang, J., Thulasiraman, K.: Polynomial time approximation algorithms for multi-constrained qos routing. *IEEE/ACM Transactions on Networking (TON)* 16(3), 656–669 (2008)

# The $k$ -Separator Problem

Walid Ben-Ameur, Mohamed-Ahmed Mohamed-Sidi, and José Neto

Institut Mines-Télécom, Télécom SudParis, CNRS Samovar UMR 5157  
9 Rue Charles Fourier, 91011 Evry Cedex, France  
`walid.benameur@it-sudparis.eu, m-ahmed.m-sidi@it-sudparis.eu,`  
`jose.neto@it-sudparis.eu`

**Abstract.** Given a vertex-weighted undirected graph  $G = (V, E, w)$  and a positive integer  $k$ , we consider the  $k$ -separator problem: it consists in finding a minimum-weight subset of vertices whose removal leads to a graph where the size of each connected component is less than or equal to  $k$ . We show that this problem can be solved in polynomial time for some graph classes: for cycles and trees by a dynamic programming approach and by using a peculiar graph transformation coupled with recent results from the literature for  $mK_2$ -free,  $(G_1, G_2, G_3, P_6)$ -free, interval-filament, asteroidal triple-free, weakly chordal, interval and circular-arc graphs. Approximation algorithms are also presented.

**Keywords:** graph partitioning, complexity theory, optimization, approximation algorithms, communication networks.

## 1 Introduction

Given a vertex-weighted undirected graph  $G = (V, E, w)$ , the minimum vertex cover problem consists in computing a minimum-weight set of vertices  $S \subset V$  such that  $V \setminus S$  is a stable set. A minimum-weight vertex cover can then be exhibited if one can find a maximum-weight stable set. While the problem can be solved in polynomial time in some cases (bipartite graphs, perfect graphs, etc.), it is known to be NP-hard in general (see, e.g., [10,17]).

Let  $k$  be a positive number. We consider the following natural generalization of the vertex cover problem. The objective is to compute a minimum-weight subset of vertices  $S$  whose removal leads to a graph where the size of each connected component is less than or equal to  $k$ . Let us call such a set a  $k$ -separator. If  $k = 1$  we get the classical vertex cover problem. The case  $k = 2$  is equivalent to computing the dissociation number of a graph (in the case of unit weights) [21]. This problem is NP-hard even if the graph is bipartite.

The  $k$ -separator problem has many applications. If vertex weights are equal to 1, the size of a minimum  $k$ -separator can be used to evaluate the robustness of a graph or a network. Intuitively, a graph for which the size of the minimum  $k$ -separator is large, is more robust. Unlike the classical robustness measure given by the connectivity, the new one seems to avoid to underestimate robustness when there are only some local weaknesses in the graph. Consider for example

a graph containing a complete subgraph and a vertex connected to exactly one vertex of the subgraph. Then the vertex-connectivity of this graph is 1 while the graph seems to be robust everywhere except in the neighborhood of one vertex. The size of a minimum  $k$ -separator of this graph is  $|V| - 1 - k$ .

The minimum  $k$ -separator problem has some other network applications. A classical problem consists in partitioning a graph/network into different subgraphs with respect to different criteria. For example, in the context of social networks, many approaches are proposed to detect communities. By solving a minimum  $k$ -separator problem, we get different connected components that may represent communities. The  $k$ -separator vertices represent persons making connections between communities. The  $k$ -separator problem can then be seen as a special partitioning/clustering graph problem.

Computing a  $k$ -separator can also be useful to build algorithms based on divide-and-conquer approaches. In some cases, a problem defined on a graph can be decomposed into many subproblems on smaller subgraphs obtained by the deletion of a  $k$ -separator (see e.g., [18]).

The  $k$ -separator problem is closely related to the vertex-separator problem where we aim to remove a minimum-weight set of vertices such that each connected component in the remaining graph has a size less than  $\alpha|V|$  (for a fixed  $\alpha < 1$ ). A polyhedral study of this problem is proposed in [1] (see also the references therein). When the vertex-separator problem is considered, the graph is generally partitioned into 3 subgraphs: the separator, and two subgraphs each of size less than  $\alpha|V|$ . The philosophy is different in the case of the  $k$ -separator where the graph is partitioned into many components each of size less than  $k$ .

The  $k$ -separator problem was considered in one published paper [14] where it was presented as a problem of disconnecting graphs by removing vertices. An extended formulation is proposed in [14] with some polyhedral results. Some other applications were also mentioned in [14]. This includes a constraint matrix decomposition application where each row  $A_i$  of a matrix  $A$  is represented by a vertex  $v_i$  and two vertices  $v_i$  and  $v_j$  are adjacent if there is at least one column  $h$  with nonzero coefficients in the corresponding two rows ( $a_{ih} \neq 0$  and  $a_{jh} \neq 0$ ). The problem is to assign as many rows as possible to so-called blocks such that no more than  $k$  rows are assigned to the same block, and rows assigned to different blocks are not connected (i.e., there is not any column  $h$  such that  $a_{ih}a_{jh} \neq 0$  if  $A_i$  and  $A_j$  are in different blocks) [3]. This matrix decomposition may help the solution process of linear or integer programs where the constraint matrix is defined by  $A$ . Another reported application is related to the field of group technology (see [14] for details).

The paper is organized as follows. Some notation is presented in Section 2. In Section 3 we discuss on cases for which we show the  $k$ -separator problem can be solved in polynomial time. We firstly deal (Subsection 3.1) with the case of trees and cycles. A particular graph transformation connecting the  $k$ -separator problem to the stable set problem is then presented (Subsection 3.2). Investigations on its structural properties coupled with recent results from the literature allowed us to show the polynomial-time solvability of the  $k$ -separator

problem and to provide methods to deal with the cases when the graph is of type:  $mK_2$ -free,  $(G_1, G_2, G_3, P_6)$ -free, interval-filament, asteroidal triple-free, weakly chordal, interval or circular-arc. Approximations algorithms are then described in Section 4.

## 2 Notation

Given a graph  $G = (V, E)$  and a vertex subset  $U \subset V$ , the complement of  $U$  in  $G$ , i.e. the vertex set  $V \setminus U$  is denoted  $\overline{U}$ . Given a vertex subset  $S \subset V$ , the set of vertices in  $\overline{S}$  that are adjacent to at least one vertex in  $S$  is denoted  $N(S)$ . Given a subset of vertices  $S \subset V$ ,  $\chi^{(S)} \in \{0, 1\}^n$  denotes the incidence vector of  $S$ , with  $n = |V|$ . The convex hull of all the incidence vectors of  $k$ -separators in the graph  $G$  is denoted  $\mathcal{S}_k(G)$ . We also use  $G(S)$  to denote the subgraph of  $G$  that is induced by a subset of vertices  $S \subset V$ .

The size of a graph denotes its number of vertices.  $K_n$  denotes a complete graph with size  $n$ . Given some integer  $m$ ,  $mK_2$  denotes a matching with  $m$  edges.

If  $G$  does not have a graph  $H$  as an induced subgraph, then we say that  $G$  is  $H$ -free.

If  $G$  is a path with vertex set  $\{v_1, \dots, v_n\}$  and edge set  $\{(v_i, v_{i+1}) : i = 1, \dots, n-1\}$ , then the notation  $[v_i, v_j]$  (resp.  $]v_i, v_j[$ ,  $[v_i, v_j[, ]v_i, v_j]$ ) with  $i < j$ ,  $i, j \in \{1, \dots, n\}$  stands for the vertex set  $\{v_i, v_{i+1}, \dots, v_j\}$  (resp.  $\{v_{i+1}, \dots, v_{j-1}\}$ ,  $\{v_i, v_{i+1}, \dots, v_{j-1}\}$ ,  $\{v_{i+1}, \dots, v_j\}$ ). The set of all the simple paths joining  $i$  and  $j$  will be denoted  $P_{ij}$ . Given a simple path  $p$  joining  $i$  and  $j$ ,  $x(p)$  stands for the sum of the  $x_v$  values over all vertices belonging to  $p$  (including  $i$  and  $j$ ).

## 3 Polynomial Cases

### 3.1 Trees and Cycles

Due to space limitations, the description of a dynamic programming algorithm in order to solve the  $k$ -separator problem when the graph is a tree or a cycle will appear in another paper. It directly leads to the following result.

**Proposition 1.** *The  $k$ -separator problem can be solved in polynomial time for trees and cycles. This holds even if  $k$  is part of the input.*  $\square$

### 3.2 Further Polynomiality Results from Connections with the Stable Set Problem

#### From $k$ -Separators to Stable Sets ... and Conversely

We now present a construction to reduce the  $k$ -separator problem to a maximum weight stable set problem.

Given a vertex-weighted graph  $G$ , we build a vertex-weighted extended graph  $G^* = (V^*, E^*, w)$  as follows. Each subset of vertices  $S \subset V$  such that  $1 \leq |S| \leq k$  and  $G(S)$  is connected, is represented by a vertex in  $G^*$ . In others words,

$V^* = \{S \subset V, |S| \leq k, G(S) \text{ is connected}\}$ . The set of edges is defined as follows:  $E^* = \{(S, T), S \in V^*, T \in V^*, S \neq T, \text{ such that either } S \cap T \neq \emptyset, \text{ or } (u, v) \in E \text{ for some } u \in S \text{ and } v \in T\}$ . Said another way,  $S \in V^*$  and  $T \in V^*$  are connected by an edge if the subsets of vertices of  $G$  they are representing either have a common vertex or contain two adjacent vertices. The weight of a vertex  $S \in V^*$  is defined by  $w_S = \sum_{v \in S} w_v$ .

Let  $R$  be a maximum-weight stable set of  $G^*$ . If two vertices  $S \in V^*$  and  $T \in V^*$  belong to this stable set  $R$ , then  $S \cap T = \emptyset$  and there are no edges in  $G$  with one endvertex in  $S$  and another endvertex in  $T$ . In other words, if we consider  $\cup_{S \in R} S$ , we get a set of vertices in  $V$  inducing a subgraph where each connected component has a size less than or equal to  $k$ . The complementary set of  $\cup_{S \in R} S$  in  $V$  is a  $k$ -separator for the graph  $G$ . This graph construction can be seen as a generalization of a construction proposed by [12] for the dissociation problem ( $k = 2$ ).

Let us now illustrate the fact that the graph transformation we just described may be used to derive interesting formulations for the  $k$ -separator problem. Hereafter is an initial formulation which straightforwardly follows from the original problem statement: the objective is to find a set of vertices with minimum cost such that it intersects each set of vertices inducing a connected component having size  $k + 1$  in the original graph. The boolean variable  $x_v$  represents the fact that the vertex  $v$  belongs ( $x_v = 1$ ) or not ( $x_v = 0$ ) to the  $k$ -separator.

$$(IP1) \begin{cases} \min \sum_{v \in V} w_v x_v \\ \sum_{v \in S} x_v \geq 1, \forall S \subset V, |S| = k + 1, G(S) \text{ connected}, \\ x_v \in \{0, 1\}, \forall v \in V. \end{cases}$$

Now, from the graph transformation above we can easily derive another formulation that we may express as follows: find a stable set in  $G^*$  (this corresponds to the set of variables  $(y_S)_{S \in V^*}$  and the constraints  $y_S + y_T \leq 1, \forall (S, T) \in E^*$ ), so that the set of nodes in  $V$  that are not represented in this stable set (i.e. a  $k$ -separator in  $G$ , from our discussion above) has minimum total cost. The boolean variable  $y_S, S \in V^*$ , represents the fact that the node  $S \in V^*$  in  $G^*$  either belongs to the stable set ( $y_S = 1$ ) or not ( $y_S = 0$ ).

$$(IP2) \begin{cases} \min \sum_{v \in V} w_v x_v \\ x_v = 1 - \sum_{S \in Q_v} y_S, \forall v \in V, \\ y_S + y_T \leq 1, \forall (S, T) \in E^*, \\ y_S \in \{0, 1\}, \forall S \in V^*, \end{cases}$$

with  $Q_v = \{T \in V^* : v \in T\}$ . Let  $F1$  (resp.  $F2$ ) stand for the set of feasible solutions of the linear relaxation of  $(IP1)$  (resp.  $(IP2)$ ) with respect to variables  $(x_v)_{v \in V}$ .

**Proposition 2.** *The following inclusion holds:  $F2 \subseteq F1$ .*

*Proof.* Let  $(x, y)$  stand for a feasible solution of the linear relaxation of  $(IP2)$ . Let  $C$  denote a connected component of size  $k+1$  in the original graph  $G = (V, E)$ . So

we have:  $\sum_{v \in C} x_v = k + 1 - \sum_{v \in C} \sum_{S \in Q_v} y_S$ . Notice that in the last expression each variable  $y_S$  such that  $S$  has a nonempty intersection with  $C$  occurs exactly  $|S \cap C|$  times.

Let  $T$  denote a spanning tree of  $C$  (in the original graph) and consider the following quantity:  $\sum_{(v,w) \in T} \sum_{S \in Q_v \cup Q_w} y_S$ . Notice that in the last expression, the number of times a variable  $y_S$  occurs is equal to the number of edges of  $T$  that intersect with  $S$ , and thus is larger than or equal to  $|S \cap C|$ . From this we deduce that  $\sum_{v \in C} \sum_{S \in Q_v} y_S \leq \sum_{(v,w) \in T} \sum_{S \in Q_v \cup Q_w} y_S$ . Moreover, using the feasibility of  $(x, y)$ , we can write that  $\sum_{(v,w) \in T} \sum_{S \in Q_v \cup Q_w} y_S \leq k$ .

Combining the two previous inequalities leads to  $\sum_{v \in C} \sum_{S \in Q_v} y_S \leq k$ . Consequently, inequality  $\sum_{v \in C} x_v \geq 1$  holds. In other words,  $x$  is a feasible solution of the linear relaxation of (IP1).  $\square$

Both linear relaxations of (IP1) and (IP2) may be easily strengthened with many families of inequalities. Such strengthened formulations and some others are under study and computational experiments to compare them (w.r.t. quality of approximation and computing times, notably) are being carried out. However those investigations go beyond the scope of this article and will be the topic of another paper. In the rest of this section we show the  $k$ -separator problem is polynomial-time solvable for many graph classes.

### **$mK_2$ -Free Graphs**

Assume that the graph  $G$  does not contain an induced matching of size  $m$  where  $m$  is a constant, i.e.  $G$  is  $mK_2$ -free.

It is shown in [15] that the dissociation problem is easy to solve in this case. Remember that the latter is equivalent to the  $k$ -separator problem with  $k = 2$ . We generalize this result for any constant  $k$ .

**Proposition 3.** *The  $k$ -separator problem can be solved in polynomial time for  $mK_2$ -free graphs if we assume that  $m$  and  $k$  are constants.*

*Proof.* Consider the extended graph  $G^*$ . Since  $k$  is a constant,  $G^*$  has a polynomial size. We know from [2] that the stable set problem can be solved in polynomial time if the graph is  $mK_2$ -free. It is then enough to prove that  $G^*$  is  $mK_2$ -free if  $G$  is  $mK_2$ -free.

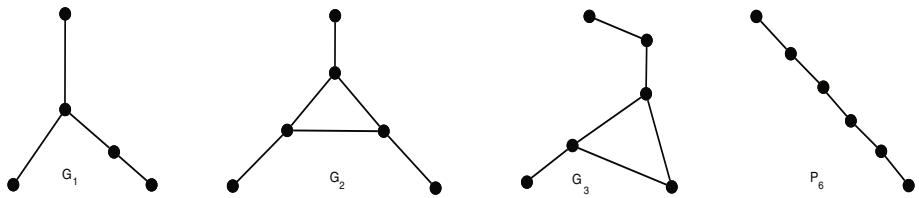
Suppose that  $G^*$  contains an induced matching of size  $m$ . Consider an edge  $(u, w)$  of  $E^*$ . Remember that  $u$  (resp.  $w$ ) represents a subset  $V_u$  (resp.  $V_w$ ) of vertices of  $V$  of size less than  $k$  such that  $G(V_u)$  (resp.  $G(V_w)$ ) is connected. Suppose that either  $|V_u| > 1$  or  $|V_w| > 1$ , then by connectivity of  $G(V_u)$  and  $G(V_w)$ , there is an edge in  $G$  connecting two vertices belonging to  $V_u \cup V_w$ . If  $|V_u| = 1$  and  $|V_w| = 1$ , then the unique vertex in  $V_u$  is clearly adjacent to the unique vertex in  $V_w$  since  $u$  and  $w$  are adjacent in  $G^*$ . In other words, there is always at least one edge connecting two vertices of  $V_u \cup V_w$ .

Moreover, given two edges of the induced matching  $(u_1, w_1)$  and  $(u_2, w_2)$ , then each vertex in  $V_{u_1} \cup V_{w_1}$  is not adjacent to any vertex in  $V_{u_2} \cup V_{w_2}$  (otherwise, the matching is not an induced one). Consequently, the graph  $G$  contains an induced matching of size  $m$ . This terminates the proof.  $\square$

### $(G_1, G_2, G_3, P_6)$ -Free Graphs

Let  $G_1$  be the chair graph (or fork) obtained from the claw by a single subdivision of one of its edges.  $G_1$  is represented on the left of Figure 1. It is proved in [11] that the maximum weight stable set problem can be solved in polynomial time if the graph is  $G_1$ -free. Their result is an improvement of the classical result of [16,13] related to claw-free graphs since the class of  $G_1$ -free graphs includes the class of claw-free graphs.

When  $k = 2$ , it is proved in [15] that the graph  $G^*$  is  $G_1$ -free if and only if  $G$  is  $(G_1, G_2, G_3)$ -free where  $G_2$  and  $G_3$  are shown on Figure 1. Formally,  $G_2 = (V_2, E_2)$ , with  $V_2 = \{1, 2, \dots, 6\}$ ,  $E_2 = \{12, 13, 23, 14, 25, 36\}$  and  $G_3 = (V_3, E_3)$ , with  $V_3 = \{1, 2, \dots, 6\}$ ,  $E_3 = \{12, 13, 23, 24, 15, 56\}$ . In the following proposition we extend this result when  $k \geq 3$ . Its proof is technical and, due to space limitations, it will be reported in another paper. The graphs  $G_1$ ,  $G_2$ ,  $G_3$ , and  $P_6$  are represented in Figure 1.



**Fig. 1.** The graphs  $G_1$ ,  $G_2$ ,  $G_3$  and  $P_6$

**Proposition 4.** *Assuming that  $k \geq 3$ , the extended graph  $G^*$  is  $G_1$ -free if and only if the original graph  $G$  is  $(G_1, G_2, G_3, P_6)$ -free.*  $\square$

**Corollary 1.** *Assuming that  $k$  is a constant  $\geq 3$ , the  $k$ -separator problem can be solved in polynomial time for  $(G_1, G_2, G_3, P_6)$ -free graphs.*

*Proof.* If  $k$  is a constant, then  $G^*$  has a polynomial size. Using Proposition 4 and the algorithm of [11] to compute a maximum weight stable set problem, one can solve the  $k$ -separator problem in polynomial time.  $\square$

### Interval-Filament, Asteroidal Triple-Free and Weakly Chordal Graphs

The results of this section are a direct consequence of the results of [5]. Given a graph  $G$  and a family  $\mathcal{H}$  of fixed connected graphs, a  $\mathcal{H}$ -packing of  $G$  is a pairwise vertex-disjoint set of subgraphs of  $G$ , each isomorphic to a member of  $\mathcal{H}$  [5]. If we add the requirement that each two subgraphs of the packing are not joined by edges, we get independent  $\mathcal{H}$ -packings. To study this problem, a graph  $\mathcal{H}(G)$  is introduced in [5]. Each subgraph of  $G$  which is isomorphic to a member of  $\mathcal{H}$  is represented by a vertex of  $\mathcal{H}(G)$ , and two vertices are adjacent if the two subgraphs either intersect or are joined by an edge.

Consider a collection of intervals on a line  $L$ . Suppose that for each interval, we are given a curve above the line, connecting the endpoints of the interval, and remaining within the limits of the interval. An interval-filament graph is the intersection graph of such a collection of intervals [6]. Computing a maximum weight stable set in interval-filament graph can be done in polynomial time [6]. It is proved in [5] that if  $G$  is an interval-filament graph, then  $\mathcal{H}(G)$  is also an interval-filament graph. In other words, the class of interval-filament graphs is closed under the operation  $G \rightarrow \mathcal{H}(G)$ . Notice that the class of interval-filament graphs includes polygon-circle graphs and cocomparability graphs.

The same was also proved in [5] for the class of weakly chordal graphs [9] (graphs such that neither the graph nor its complement contain an induced cycle on 5 or more vertices) and the class of asteroidal triple-free graphs (graphs not containing an asteroidal triple defined as a stable set of 3 vertices such that between each pair of vertices of this triple, there is path connecting them and avoiding the neighborhood of the third vertex). We know from [4] that the maximum weight stable set problem can be solved in polynomial time for asteroidal triple-free graphs. The same holds for weakly chordal graphs (see, e.g., [19]).

Let us now go back to the  $k$ -separator problem and let us slightly change the definition of  $\mathcal{H}$  by allowing it to depend on  $G$ . More precisely, let  $\mathcal{H}$  be the set of all connected subgraphs of  $G$  containing at most  $k$  vertices. Then,  $\mathcal{H}(G)$  is exactly the graph  $G^*$ . Consequently, the results of [5] can be directly applied here to deduce that the problem is easy to solve. We only have to ensure that the size of  $G^* = \mathcal{H}(G)$  is polynomially bounded. This of course occurs if  $k$  is a constant.

**Proposition 5.** *Assuming that  $k$  is a constant, the  $k$ -separator problem can be solved in polynomial time for interval-filament, asteroidal triple-free and weakly chordal graphs.*  $\square$

### Interval and Circular-arc Graphs

Interval graphs are graphs where a vertex corresponds to an interval and an edge  $(u, v)$  exists if there is a nonempty intersection between the intervals represented by  $u$  and  $v$ . We prove below that the  $k$ -separator problem is easy to solve for interval graphs.

Since interval graphs are interval-filament and chordal graphs the results of Section 3.2 can be applied here to deduce that the  $k$ -separator problem can be solved in polynomial-time for this class of graphs. However, in Section 3.2,  $k$  is required to be constant. This was necessary to get a graph  $G^*$  with a polynomial size. We prove in this section that the problem is easy to solve even if  $k$  is part of the input.

Given a graph  $G$ , one can check in linear time if the graph is an interval graph and provide a family  $\mathcal{I}$  of intervals such the graph is the intersection graph of the family [8]. We can obviously assume that for each pair of intervals  $[a, b]$  and  $[c, d]$  of  $\mathcal{I}$ , the endpoints are different ( $a \neq b \neq c \neq d$ ). Let  $[a_w, b_w]$  be the interval related to vertex  $w \in V$ . Then,  $\mathcal{I} = \{[a_w, b_w] : w \in V\}$ .

When  $G^*$  is built, the number of vertices can be non-polynomial. However, since each vertex  $v^*$  of  $G^*$  corresponds to a connected graph of  $G$ , and each vertex  $w$  of  $G$  corresponds to an interval, one can associate to  $v^*$  the union of the intervals  $\cup_{w \in v^*} [a_w, b_w]$ . The connectivity of the subgraph related to  $v^*$  clearly implies that  $\cup_{w \in v^*} [a_w, b_w]$  is an interval. Two vertices  $v^*$  and  $u^*$  are adjacent in  $G^*$  if and only if the two intervals associated with  $v^*$  and  $u^*$  intersect:  $\cup_{w \in v^*} [a_w, b_w] \cap \cup_{w \in u^*} [a_w, b_w] \neq \emptyset$ .

While the number of vertices of  $G^*$  can be non polynomial, the number of intervals that can be obtained as a union of intervals of  $\mathcal{I}$  is polynomial (quadratic). In other words, for an interval  $[x, y]$  where  $x$  and  $y$  belong to  $\cup_{w \in V} \{a_w, b_w\}$ , we might have many vertices  $v^*$  for which  $\cup_{w \in v^*} [a_w, b_w] = [x, y]$ . However, a stable set in  $G^*$  cannot simultaneously contain  $v^*$  and  $u^*$  if  $\cup_{w \in v^*} [a_w, b_w] = \cup_{w \in u^*} [a_w, b_w]$  since  $u^*$  and  $v^*$  are adjacent in  $G^*$ .

It becomes now clear that instead of building  $G^*$ , we should consider a more restricted graph  $G^{**}$ , where all vertices  $v^*$  having the same  $\cup_{w \in v^*} [a_w, b_w] = [x, y]$  are represented by only one vertex  $v_{[x,y]}$ . Two vertices  $v_{[x,y]}$  and  $v_{[a,b]}$  are adjacent if  $[x, y]$  and  $[a, b]$  intersect. The graph  $G^{**}$  obviously has a polynomial size.

In order to transform the maximum weight stable set problem in  $G^*$  into a maximum weight stable set problem in  $G^{**}$ , we have to define the weight of a vertex  $v_{[x,y]}$  of  $G^{**}$ .

Observe that  $v_{[x,y]}$  exists if the interval  $[x, y]$  is the exact union of at most  $k$  intervals of  $\mathcal{I}$ . Since a weight  $w_v$  is associated with each interval  $[a_v, b_v] \in \mathcal{I}$ , the weight of  $v_{[x,y]}$  is given by the maximum weight of at most  $k$  intervals of  $\mathcal{I}$  whose union is equal to  $[x, y]$ . More precisely, for each interval  $[x, y]$  where  $x$  and  $y$  belong to  $\cup_{w \in V} \{a_w, b_w\}$ , we should solve the problem

$$\max_{\substack{A \subset V : |A| \leq k, \\ [x,y] = \bigcup_{v \in A} [a_v, b_v]}} \sum_{v \in A} w_v. \quad (1)$$

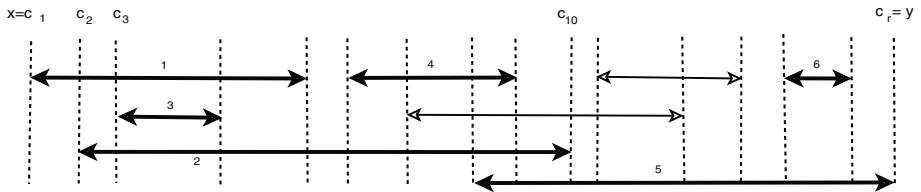
If (1) does not have a solution, then  $[x, y]$  is not represented by a vertex in  $G^{**}$ . Otherwise, the weight of  $v_{[x,y]}$  is equal to the maximum objective value of (1). We show below that (1) can be solved in polynomial time. For an interval  $[a, b] \in \mathcal{I}$ , we will use  $w_{[a,b]}$  to denote the weight  $w_v$  of the vertex  $v \in V$  representing this interval.

**Lemma 1.** *Problem (1) can be solved in polynomial time by dynamic programming.*

*Proof.* First, observe that all the intervals  $[a_v, b_v] \in \mathcal{I}$  that are not included in  $[x, y]$  can be eliminated when we are solving (1). Let  $S = \{c_1 = x, c_2, \dots, c_r = y\}$  be the set of endpoints of the intervals included in  $[x, y]$ :  $S = \bigcup_{\substack{v \in V : \\ [a_v, b_v] \subset [x, y]}} \{a_v, b_v\}$ . The sequence  $(c_i)_{1 \leq i \leq r}$  is an increasing one. Notice that we can assume that  $c_1 = x$  and  $c_r = y$ , since otherwise problem (1) does not have a solution.

The cardinality of  $S$  denoted by  $r$  is of course less than  $2|V|$ . Let  $O \subset \{1, \dots, r\}$  be the subset of indexes  $j$  such that there exists  $v \in V$  satisfying  $[a_v, b_v] \subset [x, y]$  and  $a_v = c_j$ . In this case, let  $j + \delta(j)$  be the index such that  $b_v = c_{j+\delta(j)}$ . Thus, if  $j \in O$ , then  $1 \leq \delta(j) \leq r - j$ .

Figure 2 illustrates the definitions where we have  $r = 16$ ,  $O = \{1, 2, 3, 6, 7, 8, 11, 14\}$ ,  $\delta(1) = 4$ ,  $\delta(2) = 8$ ,  $\delta(3) = 1$ ,  $\delta(6) = 3$ , etc. Assume that  $k = 6$  and suppose that the optimal solution of problem (1) is given by the intervals represented by thick arrows in Figure 2. The intervals belonging to the optimal solution can be numbered according to the order of their starting points. In Figure 2, they are numbered from 1 to 6. Let us, for example, consider the 3 first intervals belonging to the optimal solution. According to Figure 2, these 3 intervals cover the interval  $[c_1, c_{10}]$ . To reach  $y = c_{16}$ , it is clear that we should at least cover the interval  $[c_{10}, c_{16}]$  using intervals starting after  $c_4$  (because the third interval starts at  $c_3$ ). Since we already used 3 intervals to reach  $c_{10}$ , we should use at most  $k - 3 = 3$  intervals to reach  $y = c_{16}$ . Intervals numbered from 4 to 6 necessarily constitute an optimal solution of the problem that consists in covering  $[c_{10}, c_{16}]$  by no more than 3 intervals starting after  $c_4$ .



**Fig. 2.** On the dynamic programming approach to solve problem (1)

The simple observation made above directly leads to a dynamic programming approach. To make things more precise, let us introduce further notation. Let  $i_0$  and  $i_1$  be two integer numbers such that  $1 \leq i_0 \leq i_1 \leq r$ . For any integer number  $1 \leq l \leq k$ , let  $f(i_0, i_1, l)$  be the maximum weight that we can have to cover the interval  $[c_{i_1}, y]$  using at most  $l$  intervals among those starting after  $c_{i_0}$ , i.e., the set of intervals  $\{(a_w, b_w) : w \in V, c_{i_0} \leq a_w, b_w \leq y\}$ .

If  $i_1 < r$ , it is clear that to cover  $[c_{i_1}, y]$ , we need at least one interval belonging to  $\{(a_w, b_w) : w \in V, c_{i_0} \leq a_w, b_w \leq y\}$ . This clearly leads to the following induction formula:

$$f(i_0, i_1 < r, l) = \max_{j \in O: i_0 \leq j \leq i_1} w_{[c_j, c_{j+\delta(j)}]} + f(j+1, \max(j+\delta(j), i_1), l-1). \quad (2)$$

If  $O \cap \{i_0, i_0 + 1, \dots, i_1 < r\} = \emptyset$ , then  $f(i_0, i_1 < r, l) = -\infty$ . If  $l = 0$ , we also have  $f(i_0, i_1 < r, 0) = -\infty$ .

If  $i_1 = r$ , then  $y = c_r$  is already reached. The induction formula is then given by:

$$f(i_0, r, l) = \max \left( 0, \max_{j \in O: i_0 \leq j \leq r} w_{[c_j, c_{j+\delta(j)}]} + f(j+1, r, l-1) \right). \quad (3)$$

Problem (1) is solved by computing  $f(1, 1, k)$ . The complexity of the dynamic programming algorithm is obviously given by  $O(kn^3)$ .  $\square$

**Proposition 6.** *The  $k$ -separator problem can be solved in polynomial time for interval graphs. This holds even if  $k$  is not constant.*

*Proof.* We already observed that the size of the graph  $G^{**}$  is polynomially bounded. Since problem (1) can be solved in polynomial time, the weight of each vertex of  $G^{**}$  is easy to compute. Then, we only have to solve the maximum weight stable set problem in  $G^{**}$ . Using the fact that this problem is easy to solve for interval graphs, the proof is complete.  $\square$

Circular-arc graphs are a simple generalization of interval graphs. They are defined by the intersection graphs of a set of arcs on the circle. The previous proposition and the algorithm described in the proof of Lemma 1 can be generalized in an obvious way.

**Proposition 7.** *The  $k$ -separator problem can be solved in polynomial time for arc-circular graphs. This holds even if  $k$  is not constant.*  $\square$

## 4 Approximation Algorithms

The first approximation algorithm we give relies on the linear relaxation ( $LP1$ ) of the integer program ( $IP1$ ) introduced in Section 3.2. Notice that the separation of inequalities “ $\sum_{v \in S} x_v \geq 1, \forall S \subset V, |S| = k + 1, G(S)\text{connected}$ ” in ( $IP1$ ) is NP-hard even if all vertex-weights belong to  $\{0, 1\}$  when  $k$  is part of the input [7]. If  $k$  is constant, the separation is obviously easy. It is also known that a maximum-weight connected subgraph of size  $k + 1$  can be computed easily if the graph is a tree [7].

An LP-based approximation algorithm (Algorithm 1) is obtained by generalizing the basic approximation algorithm for the vertex cover problem.

A connected subgraph  $G(S)$  is said to be *large* if  $|S| \geq k + 1$ .

---

### Algorithm 1. LP-based Approximation Algorithm

---

- 1: **Input:** A vertex-weighted undirected graph  $G = (V, E, w)$  and an integer  $k$ .
  - 2: **Output:** A  $k$ -separator  $S$ .
  - 3: Solve ( $LP1$ ) and let  $x$  be an optimal solution of ( $LP1$ ).
  - 4: Set  $S := \emptyset$ .
  - 5: **while**  $G(V \setminus S)$  contains large connected components **do**
  - 6:     Select  $R \subset V \setminus S$  such that  $|R| = k + 1$  and  $G(R)$  connected.
  - 7:     Select  $v \in R$  such that  $x_v$  is maximum and set  $S := S \cup \{v\}$ .
  - 8: **end while**
- 

**Proposition 8.** *The LP-based approximation algorithm (Algorithm 1) is a  $(k + 1)$ -approximation algorithm.*

*Proof.* Since  $\sum_{y \in R} x_y \geq 1$  for each subset  $R \subset V \setminus S$  where  $|R| = k + 1$  and  $G(R)$  is connected, the vertex  $v$  (maximizing  $x_v$  inside  $R$ ) necessarily satisfies  $x_v \geq \frac{1}{k+1}$ . Adding  $v$  to  $S$  is equivalent to rounding  $x_v$  to 1. The final solution is clearly a  $k$ -separator. The weight of this  $k$ -separator is not more than  $k + 1$  times the weight of the fractional solution  $x$  (since in the rounding procedure,  $x_v$  is multiplied by at most  $k + 1$ ). Since the weight of the fractional solution  $x$  is a lower bound of the optimal weight, we deduce that we have a  $(k + 1)$ -approximation.  $\square$

Observe that the algorithm described above is a polynomial-time algorithm if we assume that  $k$  is bounded by a constant. This is necessary to guarantee that the size of  $(LP1)$  is polynomial. The primal-dual approach (see, .e.g., [20]) leads also to a  $(k + 1)$ -approximation. In fact, the  $k$ -separator problem is a special-case of the hitting set problem where we want to hit large connected components.

If all vertex weights are equal to 1, then there is another simple  $(k + 1)$ -approximation algorithm (Algorithm 2).

---

**Algorithm 2.** Greedy Approximation Algorithm

---

- 1: **Input:** A graph  $G = (V, E)$  and an integer  $k$ .
  - 2: **Output:** A  $k$ -separator  $S$ .
  - 3: Set  $S := \emptyset$ .
  - 4: **while**  $G(V \setminus S)$  contains large connected components **do**
  - 5:     Select  $R \subset V \setminus S$  such that  $|R| = k + 1$  and  $G(R)$  is connected.
  - 6:      $S := S \cup R$ .
  - 7: **end while**
- 

**Proposition 9.** *For the case when all vertex weights are equal to 1, the greedy algorithm (Algorithm 2) is a  $(k + 1)$ -approximation algorithm for the  $k$ -separator problem.*  $\square$

The greedy algorithm obviously has a polynomial time complexity even if  $k$  is part of the input.

## References

1. Balas, E., de Souza, C.: The vertex separator problem: a polyhedral investigation. Mathematical Programming 103, 583–608 (2005)
2. Balas, E., Yu, C.: On graphs with polynomially solvable maximum-weight clique problem. Networks 19, 247–253 (1989)
3. Borndörfer, R., Ferreira, C.E., Martin, A.: Decomposing matrices into blocks. SIAM Journal on Optimization 9, 236–269 (1998)
4. Broersma, H., Kloks, T., Kratsch, D., Müller, H.: Independent sets in asteroidal triple-free graphs. SIAM Journal on Discrete Math. 12, 276–287 (1999)
5. Cameron, K., Hell, P.: Independent packings in structured graphs. Mathematical Programming, Ser. B 105, 201–213 (2006)

6. Gavril, F.: Maximum weight independent sets and cliques in intersection graphs of filaments. *Information Processing Letters* 73, 181–188 (2000)
7. Goldschmidt, O., Hochbaum, D.: K-Edge Subgraphs Problems. *Discrete Applied Mathematics* 74, 159–169 (1997)
8. Habib, M., McConnell, R., Paul, C., Viennot, L.: Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition, and consecutive ones testing. *Theor. Comput. Sci.* 234, 59–84 (2000)
9. Hayard, R.B.: Weakly triangulated graphs. *Journal of Combinatorial Theory, Ser. B* 39, 200–209 (1985)
10. Korte, B., Vygen, J.: *Combinatorial optimization: theory and algorithms*. Springer (2005)
11. Lozin, V., Milanic, M.: A polynomial algorithm to find an independent set of maximum weight in a fork-free graph. *Journal of Discrete Algorithms* 6, 595–604 (2008)
12. Lozin, V., Rautenbach, D.: Some results on graphs without long induced paths. *Inform. Process. Lett.* 88, 167–171 (2003)
13. Minty, G.: On maximal independent sets of vertices in claw-free graphs. *Journal of Combinatorial Theory, Ser. B* 28, 284–304 (1980)
14. Oosten, M., Rutten, J., Spiksma, F.: Disconnecting graphs by removing vertices: a polyhedral approach. *Statistica Neerlandica* 61, 35–60 (2007)
15. Orlovich, Y., Dolgui, A., Finke, G., Gordon, V., Werner, F.: The complexity of dissociation set problems in graphs. *Discrete Applied Mathematics* 159, 1352–1366 (2011)
16. Sbihi, N.: Algorithme de recherche d'un stable de cardinalité maximum dans un graphe sans étoile. *Discrete Mathematics* 29, 53–76 (1980)
17. Schrijver, A.: *Combinatorial optimization: polyhedra and efficiency*. Springer (2003)
18. Shmoys, D.: Cut problems and their application to divide-and-conquer. In: Hochbaum, D.S. (ed.) *Approximation Algorithms for NP-hard Problems*, pp. 192–235. PWS Publishing (1997)
19. Spinrad, J.P., Sritharan, R.: Algorithms for weakly triangulated graphs. *Discrete Appl. Math.* 59, 181–191 (1995)
20. Williamson, D.: The primal-dual method for approximation algorithms. *Mathematical Programming, Ser. B* 91, 447–478 (2002)
21. Yannakakis, M.: Node-deletion problems on bipartite graphs. *SIAM Journal on Computing* 10, 310–327 (1981)

# On the Treewidth of Dynamic Graphs

Bernard Mans and Luke Mathieson

Department of Computing, Macquarie University, Sydney, NSW 2109, Australia  
`{bernard.mans, luke.mathieson}@mq.edu.au`

**Abstract.** *Dynamic* graph theory is a novel, growing area that deals with graphs that change over time and is of great utility in modelling modern wireless, mobile and dynamic environments. As a graph evolves, possibly arbitrarily, it is challenging to identify the graph properties that can be preserved over time and understand their respective computability.

In this paper we are concerned with the *treewidth* of dynamic graphs. We focus on *metatheorems*, which allow the generation of a series of results based on general properties of classes of structures. In graph theory two major metatheorems on treewidth provide complexity classifications by employing structural graph measures and finite model theory. Courcelle's Theorem gives a general tractability result for problems expressible in monadic second order logic on graphs of bounded treewidth, and Frick & Grohe demonstrate a similar result for first order logic and graphs of bounded local treewidth.

We extend these theorems by showing that dynamic graphs of bounded (local) treewidth where the length of time over which the graph evolves and is observed is finite and bounded can be modelled in such a way that the (local) treewidth of the underlying graph is maintained. We show the application of these results to problems in dynamic graph theory and dynamic extensions to static problems. In addition we demonstrate that certain widely used dynamic graph classes naturally have bounded local treewidth.

## 1 Introduction

Graph theory has proven to be an extremely useful tool for modelling computational systems and with the advent and growing preponderance of mobile devices and dynamic systems it is natural that graph theory is extended and adapted to capture the evolving aspects of these environments. Dynamic graphs have been formalised in a number of ways: e.g., time-varying graphs [7,8,16], carrier-based networks [4], evolving graphs [6,15,5], delay-tolerant networks [22], dynamic networks [25,24], scheduled networks [1], temporal networks [23], opportunistic networks [9,21], Markovian [10]. When considering the dynamic aspects of a dynamic graph, even classically simple properties such as shortest paths become more complex to compute and may even become definitionally ambiguous [5].

In this paper, we are not particularly interested in any particular dynamic model. Initially we will loosely use the term *dynamic graph* and for our purpose

we will define the term formally using the simplest possible definition as it is a generalized and reasonably assumption free model for a dynamic graph. For this paper, one of the key questions in moving from static graph theory to dynamic graph theory concerns the preservation of properties and their computability.

One important general approach to the complexity and computability of properties in (static) graphs is the application of *metatheorems* which classify large classes of problems. Arguably the most famous of these metatheorems is *Courcelle's theorem* (stated and proved over a series of articles from [12] to [13]) which gives a polynomial time algorithm for any monadic second-order expressible property on graphs of bounded treewidth. More precisely the model checking problem for monadic second-order logic is fixed-parameter tractable with parameter  $|\phi| + k$  where  $\phi$  is the sentence of logic and  $k$  is the treewidth of the input structure. Frick and Grohe [19] give a similar metatheorem for first-order logic and structures of locally bounded treewidth, which allows a greater class of structures (all structures of bounded treewidth have locally bounded treewidth), but constrains the logical expressibility. In fact Dvořák *et al.* [14] show that properties expressible in first-order logic are decidable in linear time for graphs of bounded expansion, a superclass of several classes of sparse graphs, including those with bounded local treewidth. Stewart [26] demonstrates that Frick and Grohe's result holds if the bound on the local treewidth is dependent on the parameter of the input problem, rather than simply constant.

Such metatheorems are extremely useful classification tools, and having them available for use in the context of dynamic graphs would be highly desirable. Two questions immediately arise when considering such an extension — are there any restrictions that have to be made to the logic and are there any further constraints on the structures (in this case the graphs)? Answering the first question is simple: no. An apparent natural match for dynamic graphs is temporal logic (a form of modal logic), however temporal logic has a simple canonical translation into first-order and monadic second-order logic. The addition of a “time” relation, that provides a temporal ordering is sufficient to capture the ideas of temporal logic. Thus we are not limited (more than before) with the logics that are applicable. With regards to the structure, we must first consider the setting of these metatheorems. They are both true in the context of finite model theory, emphasizing the *finite*. Therefore we are immediately limited to finite temporal domains (a finite domain for the vertices and hence edges of the graph is expected of course). Beyond this if we are to capture the temporal aspect in the structure, we must do so in a manner that is both useable and respects the constraints on the structure necessary for the metatheorems — namely bounded treewidth and local treewidth respectively.

Dynamic problems have been approached before in a number of ways. For instance, Hagerup [20] examines the situation where the logical expression changes (*i.e.* the question being asked changes), and Bodlaender [3], Cohen *et al.* [11] and Frederickson [18] give a variety of results that deal with graphs that change, requiring changes in the tree decompositions used. In contrast this work deals with problems that include the temporal dimension as an intrinsic aspect, rather

than problems that can be answered at an instant in time (and then may need to be recomputed). A simple example is a *journey*, the temporal extension of a path, where the route between two vertices may not exist at any given point of time, but as the graph changes the schedule of edges may allow traversal.

The remainder of this paper deals with the translation of dynamic graphs into structures that maintain these bounds if the original graph did so at every point in time. In addition we demonstrate the utility of these metatheorems in classification by application to some open problems on dynamic graphs.

## 2 Preliminaries

The graphs we employ are simple graphs, both directed and undirected. We define a *dynamic graph* for our purposes as a (di)graph augmented with a set  $T$  of times, a function  $\zeta_e$  which maps the edge set to a subset of  $T$  and a function  $\zeta_v$  that maps the vertex set to subsets of  $T$ , representing the times at which the edge or vertex exists. We will consider a discrete and finite set  $T$  of times for two main reasons: (i) typically, in most applications at hand, a continuous set  $T$  could be easily made discrete when considering time of changes of the graph as an event-based model; (ii) similarly, applications of interest are not focussed on the observation of the evolution of the graph over an infinite period, but rather on some sufficiently large, yet finite, period where some expected or asymptotic behaviour can be identified.

More formally a dynamic graph  $G = (V, E, T, \zeta_v, \zeta_e)$  consists of a set  $V$  of vertices, a set  $E \subset V \times V$  of edges (where  $vv \notin E$  for any  $v \in V$ ), a set  $T$  of times, a function  $\zeta_v : V \rightarrow 2^T$  and a function  $\zeta_e : E \rightarrow 2^T$ .

We denote the static graph derived from a dynamic graph  $G$  by taking the snapshot at time  $t$  as  $G_t$ . We do not allow edges to exist when either of their vertices do not, although the case where they do could be of interest in modelling, for example, wireless networks with long transmission ranges. We also assume that there is some order  $<_T$  defined over  $T(G)$ , however except where specified we do not assume that this order has any particular properties (*i.e.* we assume neither totality nor transitivity).

A *journey* in a dynamic graph is simply a path *over time*, that is it is a path where the edges (and vertices) of the path may not all exist at a single point in time, but where there is an ordered sequence of times such that each edge in sequence exists at a point in time after its predecessor. Note that this inherently makes a journey directed. We will denote a journey from vertex  $u$  to vertex  $v$  by  $\mathcal{J}(u, v)$ .

This dynamic graph definition is independent from other referenced definitions, though relationships can be drawn. Again, here, we aim to focus on the graph theoretic aspects, rather than modelling a particular system, thus we attempt to keep the definition as simple and open as possible. We also note that a similar definition with a continuous notion of time can be converted to an *event-based* model, which then fulfils the requirements of our definition.

### 3 Dynamic Graphs as Logical Structures

Let  $G$  be a dynamic graph. We give two translations of  $G$  into a logical structure, with different properties and limitations appropriate to different applications.

#### 3.1 Local Treewidth Preserving Structure

Let  $G = (V, E, T, \zeta_v, \zeta_e)$  be a dynamic graph.

Let  $\mathcal{A}(G)$  be the structure obtained from  $G$  with universe  $A$  and vocabulary  $\tau$  where  $A$  consists of an element  $v^t$  for each  $v \in V(G)$  if  $t \in \zeta_v(v)$ , with a function  $f_V$  such that  $f_V(v^t) = f_V(u^{t'})$  if and only if  $v^t$  and  $u^{t'}$  are derived from the same vertex  $v \in V(G)$ , an element for each  $t \in T(G)$ , and  $\tau = \{V^A, E^A, T^A, R^A, L_t^A\}$  for all  $t \in T$  where  $v \in V^{\mathcal{A}(G)}$  if and only if  $v \in V(G)$ ,  $uv \in E^{\mathcal{A}(G)}$  if and only if  $uv \in E(G)$ ,  $t \in T^{\mathcal{A}(G)}$  if and only if  $t \in T(G)$ ,  $(t_1, t_2) \in R^{\mathcal{A}(G)}$  if and only if  $t_1, t_2 \in T(G)$  and  $t_1 <_T t_2$ ,  $v \in L_t^{\mathcal{A}(G)}$  if and only if  $t \in \zeta_v(v)$ , and  $uv \in L_t^{\mathcal{A}(G)}$  if and only if  $t \in \zeta_e(uv)$ .

When the structure is understood from context, we will drop the  $\mathcal{A}$  superscript.

**Theorem 1.** *Given a dynamic graph  $G$ , if  $G_t$  has local treewidth (effectively) bounded by  $f(r)$  at each time  $t \in T(G)$ , then the Gaifman graph  $\mathcal{G}$  obtained from  $\mathcal{A}(G)$  has local treewidth (effectively) bounded by  $\max\{f(r), |T(G)| - 1\}$ .*

*Proof.* Let  $G_t$  be the graph at time  $t \in T(G)$  and  $\mathcal{G}_t$  the corresponding component of  $\mathcal{G}$ .  $\mathcal{G}_t$  is essentially  $G_t$ .

As  $G_t$  has (effectively) bounded local treewidth, there is a function  $f$  such that for each  $v \in V$  and  $r \in \mathbb{N}$  we have  $\text{tw}([N_r(v)]) \leq f(r)$  in  $G_t$  (note that this may not be true if we consider  $G$  in total, ignoring the timing of the edges). In particular for each  $v$  and  $r$  in each  $G_t$ , we have a tree decomposition where each bag has at most  $f(r) + 1$  elements. Moreover we have such a decomposition at each time  $t$ , with no interaction between any two snapshots. Therefore the width of the decomposition is at most  $f(r)$ .

The only component of  $\mathcal{G}$  that does not have an appropriate decomposition is the clique defined by the time elements representing  $T(G)$ . The only possible decomposition is to place all elements in one bag, with width  $|T(G)| - 1$ .

As each  $\mathcal{G}_t$  and the clique are disjoint, there are no additional edges to consider in the possible decompositions.  $\square$

We note also that if the graph has bounded degree at every  $t \in T(G)$ , this property is also preserved in the Gaifman graph. With the construction of Section 3.3, planarity can also be preserved.

#### 3.2 Treewidth Preserving Structure

If we are only concerned with the treewidth of the structure, we can use somewhat more natural structure.

Let  $G = (V, E, T, \zeta_v, \zeta_e)$  be a dynamic graph.

In this case the universe  $A$  consists of an element  $v^t$  for each  $v \in (V)$  if  $t \in \zeta_v(v)$ , an element for each  $t \in T(G)$ , and  $\tau = \{V^A, L_v^A, \Xi^A, T^A, R^A\}$  where  $v^t \in V^{A(G)}$  if and only if  $v \in V(G)$  and  $t \in \zeta_v(v)$ ,  $v^t \in L_v^{A(G)}$  if and only if  $v^t$  is generated from  $v$ ,  $(u, v, t) \in \Xi^{A(G)}$  if and only if  $uv \in E(G)$  and  $t \in \zeta_e(uv)$ ,  $t \in T^{A(G)}$  if and only if  $t \in T(G)$ , and  $(t_1, t_2) \in R^{A(G)}$  if and only if  $t_1, t_2 \in T(G)$  and  $t_1 <_T t_2$ .

**Theorem 2.** *If  $G$  has  $\text{tw}(G) \leq k$  at every time  $t \in T(G)$  then the Gaifman graph  $\mathcal{G}$  obtained from  $\mathcal{A}(G)$  has  $\text{tw}(\mathcal{G}) \leq \max\{k+1, |T(G)|-1\}$ .*

*Proof.* Let  $G_t$  be the snapshot of  $G$  at time  $t \in T(G)$  and let  $\text{tw}(G_t) \leq k$  for every  $t \in T(G)$ . The Gaifman graph  $\mathcal{G}$  consists of  $|T(G)| + 1$  components; a component  $\mathcal{G}_t$  for each  $G_t$  which is the normal Gaifman graph translation of a graph, i.e. the graph itself, and a ( $|T(G)|$ )-clique corresponding to the time elements of  $A$ . Each element of the time clique is connected to every vertex in exactly one  $\mathcal{G}_t$  (precisely the element corresponding to time  $t \in T(G)$ ) and each vertex in each  $\mathcal{G}_t$  is connected to exactly one vertex of the time clique.

By assumption for each  $\mathcal{G}_t$  we have  $\text{tw}(\mathcal{G}_t) \leq k$ . Thus each  $\mathcal{G}_t$  has a tree decomposition where each bag has at most  $k+1$  elements. To each of these bags we add the element corresponding to time  $t \in T(G)$ . The time clique forms a bag of size  $|T(G)|$ . We add an edge to the tree decomposition between the time clique bag and an arbitrarily chosen bag from each decomposition of each  $\mathcal{G}_t$ . Thus we have a tree decomposition of  $\mathcal{G}$  with bag size at most  $\max\{k+2, |T(G)|\}$  and therefore  $\text{tw}(\mathcal{G}) \leq \max\{k+1, |T(G)|-1\}$ .  $\square$

### 3.3 Structures for Totally Ordered Time

If we can assume that time is linear for our dynamic graph  $G$ , i.e. there exists a total order on the elements of  $T(G)$ , we may construct a structure where the Gaifman graph avoids the clique created by the elements corresponding to the times. The trade-off is that the construction of logical sentences becomes more involved in the sense that the sentences become longer.

To obtain this modified structure we restrict the relation  $R$  in each  $\tau$  such that  $(t_1, t_2) \in R$  if and only if  $t_1$  immediately precedes  $t_2$ . Furthermore we add a constant  $s$  such that  $(s, t_i) \in R$  if and only if  $t_i$  is the earliest time element. Then the component of the Gaifman graph constructed from  $\mathcal{A}(G)$  corresponding to  $R$  is a path, where each  $t_i$  subtends a vertex, and there is an edge between two such vertices if and only if one immediately precedes the other temporally. Moreover we can define the distance between each time element and  $s$  recursively:  $d_1(t_i) := Rst_i$  and  $d_n(t_i) := \exists t_j (Tt_j \wedge Rt_it_j \wedge d_{n-1}(t_j))$ .

Note of course that as first order logic is compact, these formulæ must be defined separately for each  $|T(G)|$ , giving a finite set in each case. As  $|T(G)|$  will always be taken as a parameter we effectively produce an infinite family of such formulæ, and select the appropriate subset for each instance of whichever problem we deal with. In the case of second order logic this problem does not

exist, but it is convenient to use the first order construction. Then within the logic we can define the order relation over the time elements as  $t_i \leq t_j := (t_i = t_j) \vee \bigvee_{l \in [1, |T(G)|]} (d_l(t_i) \wedge \neg d_l(t_j))$

The operator  $\leq$  allows comparison over all times, but at the cost of formulae which may depend on the size of  $T(G)$ . However there is a notable change in treewidth and local treewidth of the structures using this approach.

**Theorem 3.** *Given a dynamic graph  $G$ , let  $\mathcal{A}(G)$  be the structure constructed from  $G$  as a Local Treewidth Preserving Structure, except with a linear time relation  $R$ .*

*If  $G$  has local treewidth (effectively) bounded by  $f(r)$  at every  $t \in T(G)$ , then the Gaifman graph  $\mathcal{G}$  derived from  $\mathcal{A}(G)$  has local treewidth (effectively) bounded by  $f(r)$ .*

*Proof.* The only change to the structure in this case is the component concerned with the time elements. In the first construction this was a clique of size  $|T(G)|$ , but now is a path of length  $|T(G)|$ . As trees have treewidth 1, and local treewidth is bounded by treewidth, the local treewidth of this component is at most 1.  $\square$

**Theorem 4.** *Given a dynamic graph  $G$ , let  $\mathcal{A}(G)$  be the structure constructed from  $G$  as Treewidth Preserving Structure, except with a linear time relation  $R$ .*

*If  $G$  has  $\text{tw}(G) \leq k$  at every  $t \in T(G)$ , then the Gaifman graph  $\mathcal{G}$  derived from  $\mathcal{A}(G)$  has  $\text{tw}(\mathcal{G}) \leq k + 1$ .*

*Proof.* As before each component corresponding to the graph  $G_t$  at time  $t$  has treewidth at most  $k$ , and as before we add  $t$  to each bag of this decomposition. For the time component we construct the following tree decomposition:

1. Each vertex  $t$  is given a bag labelled  $\{t\}$ ,
2. Each edge  $st$  is given a bag labelled  $\{s, t\}$  and
3. each vertex bag is connected in the decomposition to the two edge bags that contain the same label.

This component clearly has treewidth 1. Then we complete the decomposition for  $\mathcal{G}$  by adding a decomposition edge from the vertex bag for time element  $t$  to an arbitrary bag in the decomposition of  $G_t$ . Clearly this is still a tree, and the whole decomposition has width at most  $k + 1$ .  $\square$

Note that for both representations of the temporal element of the graph, the complexity of the problem is at least partially dependent on  $|T(G)|$ , in that either the (local) treewidth depends on  $|T(G)|$ , or the length of any formula where we have to compare times does.

## 4 Applications to Dynamic Graph Problems

### 4.1 Adapting the Metatheorems to the Dynamic Context

**Theorem 5.** *Let  $G$  be a dynamic graph with  $\text{tw}(G) \leq k$  at every time  $t$  and  $\phi$  a sentence of monadic second order logic. The problem  $\text{MC}(G, \phi)$  is fixed-parameter tractable with parameter  $k + |\phi| + |T(G)|$ .*

*Proof.* Courcelle's Theorem gives us that the model checking problem for graphs of bounded treewidth is fixed-parameter tractable with the treewidth and the length of the formula as a combined parameter.

In our case we may apply Courcelle's theorem to the Gaifman graph derived from the dynamic graph. As the treewidth of the Gaifman graph is bounded by the treewidth of the original graph and  $|T(G)|$ , we obtain our result. Note that in the case where we use the linear time structure variant,  $|T(G)|$  is implicitly included in  $|\phi|$ , if necessary.  $\square$

**Theorem 6.** *Let  $G$  be a dynamic graph with effectively bounded local treewidth at every time  $t$  and  $\phi$  a sentence of first order logic. The problem  $\text{MC}(G, \phi)$  is fixed-parameter tractable with parameter  $|\phi| + |T(G)|$ .*

*Proof.* By Frick and Grohe's Theorem, the model checking problem for graphs of effectively bounded local treewidth is fixed parameter tractable with the length of the formula as the parameter. Again the length of the formula may implicitly depend upon other parameters as part of the problem.

Stewart notes that Frick and Grohe's theorem still holds if the bound on the local treewidth depends upon a parameter, rather than simply being constant. In the case where we do not use the linear time construction, the local treewidth is bounded by  $\max\{f(r), |T(G)| - 1\}$ .  $\square$

We demonstrate the use of these theorems by application to some dynamic graph problems of general interest. Bhadra and Ferreira [2] prove that the problem of finding a connected component of size at least  $k$  in an evolving digraph is  $NP$ -complete.

#### STRONGLY CONNECTED DYNAMIC COMPONENT (SCDC)

*Instance:* A dynamic digraph  $D = (V, E, T, \zeta_v, \zeta_e)$ , an integer  $k$ .

*Parameter:*  $k + |T|$ .

*Question:* Is there a set  $V' \subseteq V$  with  $|V'| \geq k$  such that for every pair  $u, v \in V'$  we have  $\mathcal{J}(u, v)$ ?

Adapting the structures for directed graphs is simply a matter of semantics for the  $E$  relation in  $\tau$ . Using this we apply the metatheorem to demonstrate that the problem is fixed-parameter tractable.

**Lemma 1.** *SCDC is expressible in first order logic.*

*Proof.* We first define a sentence that expresses the idea of a journey:

$$\begin{aligned} \mathcal{J}_n(u, v) := & \mathcal{J}_{n-1}(u, v) \vee \\ & \exists x_1, \dots, x_{n+1}, t_1, \dots, t_n (\bigwedge_{i \in [1, n+1]} Vx_i \wedge \bigwedge_{i \in [1, n]} Tt_i) \wedge \\ & (\bigwedge_{i \in [1, n]} Ex_i x_{i+1} \wedge L_{t_i} x_i x_{i+1}) \wedge \\ & (\bigwedge_{i \in [1, n]} t_i \leq t_{i+1}) \wedge (f_V(u) = f_V(x_1) \wedge f_V(v) = f_V(x_{n+1})). \end{aligned}$$

Using this the problem can be succinctly defined in first order logic as:

$$\exists v_1, \dots, v_k \forall x, y (f_V(x) = f_V(v_i) \wedge f_V(y) = f_V(v_j) \Rightarrow \mathcal{J}_k(x, y))$$

Note that the length of the sentence depends upon only  $k$  and  $|T(G)|$  and that we again avoid running afoul of the compactness of first order logic by restricting the possible journey lengths to  $k$  for each instance.  $\square$

Combining Theorem 6 and Lemma 1 we obtain the tractability result.

**Corollary 1.** SCDC is fixed-parameter tractable for dynamic graphs of bounded local treewidth.

As first order logic is a subset of monadic second order logic we may also use Theorem 5.

**Corollary 2.** SCDC is fixed-parameter tractable for dynamic graphs of bounded treewidth.

Another interesting dynamic graph problem is whether a sending vertex can receive a reply without the reply having to travel too far.

#### SHORT MESSAGE RETURN PATH (SMRP)

*Instance:* A dynamic digraph  $D = (V, E, T, \zeta_v, \zeta_e)$  with an identified vertex  $v \in V$  and an integer  $k$ .

*Parameter:*  $k + |T|$ .

*Question:* Is there a journey from each  $u \in N^{out}(v)$  to  $v$  of length at most  $k$ ?

**Lemma 2.** SMRP is expressible in first order logic.

*Proof.* Using the ability to express the idea of a journey in first order logic, SMRP is simple to express:  $\forall u (Vu \wedge Evu \Rightarrow \exists u' (f_V(u) = f_V(u') \wedge \mathcal{J}_k(u, v)))$   $\square$

Using Theorems 6 and 5 with Lemma 2 we obtain the following tractability result.

**Corollary 3.** SHORT MESSAGE RETURN PATH is fixed-parameter tractable for dynamic graphs of bounded local treewidth and bounded treewidth.

## 4.2 Transferring Static Results

The transference of Courcelle's and Frick & Grohe's metatheorems also imply the transference of previous results on static graph, with only minor changes in problem formulation. Most immediately, these results hold where we relax requirements such that the edges required for the solution exist at some point.

We demonstrate by “temporalizing”  $k$ -COLORING and the associated MSO formula that demonstrates it fixed-parameter tractability for graphs of bounded treewidth.

### PERMANENT COLORING

*Instance:* A dynamic graph  $G = (V, E, T, \zeta_v, \zeta_e)$ , an integer  $k$ .

*Parameter:*  $k + |T|$ .

*Question:* Is there a static assignment of  $k$  colors to  $V$  that is a proper coloring of  $G$  at each time  $t \in T$ ?

**Lemma 3.** PERMANENT COLORING *is expressible in monadic second order logic.*

*Proof (Sketch).*

$$\exists V_1, \dots, V_k \forall x, y, t (\bigvee_{i \in [1, k]} \bigwedge_{j \neq i} (V_i x \wedge \neg V_j x) \wedge \bigvee_{i \in [1, k]} \bigwedge_{j \neq i} (V_i y \wedge \neg V_j y) \wedge \\ Vx \wedge Vy \wedge Tt \wedge \text{Exyt} \Rightarrow \bigwedge_{i \in [1, k]} \neg(V_i x \wedge V_i y))$$

□

Alternatively we can reframe the problem as requiring a solution that is true at every point in time.

### EVOLVING COLORING

*Instance:* A dynamic graph  $G = (V, E, T, \zeta_v, \zeta_e)$ , an integer  $k$ .

*Parameter:*  $k + |T|$ .

*Question:* Is there a (possibly different) proper  $k$ -coloring of  $G$  at each time  $t \in T$ ?

**Lemma 4.** EVOLVING COLORING *is expressible in monadic second order logic.*

*Proof (Sketch).*

$$\forall t \exists V_1, \dots, V_k \forall x, y (\bigvee_{i \in [1, k]} \bigwedge_{j \neq i} (V_i x \wedge \neg V_j x) \wedge \bigvee_{i \in [1, k]} \bigwedge_{j \neq i} (V_i y \wedge \neg V_j y) \wedge \\ Vx \wedge Vy \wedge Tt \wedge \text{Exyt} \Rightarrow \bigwedge_{i \in [1, k]} \neg(V_i x \wedge V_i y))$$

□

Then by Theorem 5:

**Corollary 4.** PERMANENT COLORING and EVOLVING COLORING *are fixed parameter tractable on graphs of bounded treewidth.*

It is easy to see that, although the formulæ become somewhat more complex, it is a simple matter to translate existing expressibility results (often already in the context of parameterized tractability) into results for dynamic graphs. Thus the permanent and evolving versions of interesting problems such as DOMINATING SET, INDEPENDENT SET, VERTEX COVER, CLIQUE and SUBGRAPH ISOMORPHISM are fixed-parameter tractable on dynamic graphs of bounded (local) treewidth (Flum & Grohe [17] give appropriate static results).

## 5 Dynamic Graph Classes with Bounded (Local) Treewidth

One broad class of dynamic graphs that has proven interesting due to the connectivity problems induced are sparse dynamic graphs. There are several definitions of what constitutes a sparse graph in the dynamic context, with two common definitions involving Markovian processes and bounded degree graphs. Both of these classes in fact have bounded local treewidth.

Flum & Grohe [17] note that given a graph  $G$  of maximum degree  $d$  we have  $\text{ltw}(G, r) \leq d^r$ . Thus the class of bounded degree dynamic graphs has bounded local treewidth.

As mentioned, sparse graphs may also be defined by Markovian processes on the edges of the graph where the probability of an edge existing, and continuing to exist are bounded (e.g., [10]). This also leads to bounded local treewidth for such graphs.

**Theorem 7.** *Let  $G = (G_1, \dots, G_t)$  be a edge-Markovian dynamic graph where the probability of a non-edge becoming an edge is  $p \leq c_1/n$  and the probability of an edge becoming a non-edge is  $q \leq 1 - c_2/n$ , then the expected maximum degree for each  $G_{i \geq 2}$  is at most  $c_1 + c_2$ .*

*Proof.* At any time  $t$ , for each vertex  $v \in V(G)$  we have

$$\begin{aligned} |N_{t+1}(v)| &= (1 - q) \cdot |N_t(v)| + p(n - 1 - |N_t(v)|) \\ &\leq \frac{c_2}{n} \cdot |N_t(v)| + \frac{c_1}{n} \cdot n - \frac{c_1}{n} \cdot |N_t(v)| - \frac{c_1}{n} \\ &= c_2 \cdot \frac{|N_t(v)|}{n} + c_1 - c_1 \frac{|N_t(v)| + 1}{n} \\ &\leq c_1 + c_2 \end{aligned}$$

□

**Corollary 5.** *Let  $G = (G_1, \dots, G_t)$  be a edge-Markovian dynamic where the probability of a non-edge becoming an edge is  $p \leq c_1/n$  and the probability of an edge becoming a non-edge is  $q \leq 1 - c_2/n$ , then  $G$  is expected to have bounded local treewidth.*

We note that the third common definition for sparse dynamic graphs, using some measure of density of the graph, does not lead to bounded local treewidth. Some measures of density lead to bounded local treewidth, but essentially by implying bounded degree.

## 6 Conclusion

The structural parameters treewidth and local treewidth have proven to be very useful in algorithmic design when dealing with hard problems. Aiding in this task are the metatheorems of Courcelle and Frick & Grohe that provide complexity

classifications for large classes of graphs using logical expressibility. As dynamic graph theory and dynamic graph problems become more prominent thanks to the advance of technology, questions of treewidth arise. We have shown that given a finite, bounded temporal context the notions of logical expressibility and the corresponding metatheorems can be extended to provide tools for classification of problems on dynamic graphs. In particular if a dynamic graph has bounded (local) treewidth at every point in time, then we can construct a logical structure, and hence Gaifman graph where the (local) treewidth remains bounded. Furthermore in the finite, bounded temporal context first order and thus second order logic can be augmented (finitely) with relations that can express the temporal notion of before and after. With these tools we can easily classify many problems, including temporal extension of static problems.

A natural focus for extending this work would be to increase the graph classes that can be classified, ideally up to graphs of bounded expansion *à la* Dvořák *et al.* [14], though of course we cannot do better unless it is also possible for static graphs.

This approach however is limited by the fact that we require the length of the time interval to be finite and a parameter. If the length of time is unbounded (even finite but unbounded), then we do not have the logical expressibility to deal with these problems. Thus certain classes of dynamic graphs are excluded by this condition. However we can still easily express properties over a finite amount of time, or in periodic graphs, which covers a significant amount of real applications, where finite time is a constraint of the setting, rather than an artificial stipulation of the theory.

## References

1. Berman, K.A.: Vulnerability of scheduled networks and a generalization of Menger's theorem. *Networks* 28, 125–134 (1996)
2. Bhadra, S., Ferreira, A.: Complexity of connected components in evolving graphs and the computation of multicast trees in dynamic networks. In: Pierre, S., Barbeau, M., Kranakis, E. (eds.) ADHOC-NOW 2003. LNCS, vol. 2865, pp. 259–270. Springer, Heidelberg (2003)
3. Bodlaender, H.L.: Dynamic algorithms for graphs with treewidth 2. In: van Leeuwen, J. (ed.) WG 1993. LNCS, vol. 790, pp. 112–124. Springer, Heidelberg (1994)
4. Brejová, B., Dobrev, S., Královič, R., Vinař, T.: Routing in carrier-based mobile networks. In: Kosowski, A., Yamashita, M. (eds.) SIROCCO 2011. LNCS, vol. 6796, pp. 222–233. Springer, Heidelberg (2011)
5. Bui-Xuan, B.-M., Ferreira, A., Jarry, A.: Computing shortest, fastest, and foremost journeys in dynamic networks. *Int. J. Found. Comput. Sci.* 14(2), 267–285 (2003)
6. Casteigts, A., Chaumette, S., Ferreira, A.: Characterizing topological assumptions of distributed algorithms in dynamic networks. In: Kutten, S., Žerovnik, J. (eds.) SIROCCO 2009. LNCS, vol. 5869, pp. 126–140. Springer, Heidelberg (2010)
7. Casteigts, A., Flocchini, P., Mans, B., Santoro, N.: Deterministic computations in time-varying graphs: Broadcasting under unstructured mobility. In: Calude, C.S., Sassone, V. (eds.) TCS 2010. IFIP AICT, vol. 323, pp. 111–124. Springer, Heidelberg (2010)

8. Casteigts, A., Flocchini, P., Quattrociocchi, W., Santoro, N.: Time-varying graphs and dynamic networks. *IJPEDS* 27(5), 387–408 (2012)
9. Chaintreau, A., Mtibaa, A., Massoulié, L., Diot, C.: The diameter of opportunistic mobile networks. In: CoNEXT, p. 12 (2007)
10. Clementi, A.E.F., Monti, A., Silvestri, R.: Modelling mobility: A discrete revolution. *Ad Hoc Networks* 9(6), 998–1014 (2011)
11. Cohen, R.F., Sairam, S., Tamassia, R., Vitter, J.S.: Dynamic algorithms for optimization problems in bounded tree-width graphs. In: Rinaldi, G., Wolsey, L.A. (eds.) *Proceedings of the 3rd Integer Programming and Combinatorial Optimization Conference*, Erice, Italy, pp. 99–112. CIACO (1993)
12. Courcelle, B.: The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation* 85(1), 12–75 (1990)
13. Courcelle, B.: The monadic second-order logic of graphs xvi: Canonical graph decompositions. *Logical Methods in Computer Science* 2(2) (2006)
14. Dvořák, Z., Král, D., Thomas, R.: Deciding first-order properties for sparse graphs. In: 51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, Las Vegas, Nevada, pp. 133–142. IEEE Computer Society (2010)
15. Ferreira, A.: Building a reference combinatorial model for manets. *IEEE Network* 18(5), 24–29 (2004)
16. Flocchini, P., Mans, B., Santoro, N.: On the exploration of time-varying networks. *Theor. Comput. Sci.* 469, 53–68 (2013)
17. Flum, J., Grohe, M.: *Parameterized complexity theory*. Springer (2006)
18. Frederickson, G.N.: Maintaining regular properties dynamically in  $k$ -terminal graphs. *Algorithmica* 22(3), 330–350 (1998)
19. Frick, M., Grohe, M.: Deciding first-order properties of locally tree-decomposable structures. *Journal of the ACM* 48(6), 1184–1206 (2001)
20. Hagerup, T.: Dynamic algorithms for graphs of bounded treewidth. *Algorithmica* 27(3), 292–315 (2000)
21. Jacquet, P., Mans, B., Mühlenthaler, P., Rodolakis, G.: Opportunistic routing in wireless ad hoc networks: Upper bounds for the packet propagation speed. *IEEE Journal on Selected Areas in Communications* 27(7), 1192–1202 (2009)
22. Jacquet, P., Mans, B., Rodolakis, G.: Information Propagation Speed in Mobile and Delay Tolerant Networks. *IEEE Transactions on Information Theory* 56(1), 5001–5015 (2010)
23. Kempe, D., Kleinberg, J.M., Kumar, A.: Connectivity and inference problems for temporal networks. In: STOC, pp. 504–513 (2000)
24. Kuhn, F., Lynch, N.A., Oshman, R.: Distributed computation in dynamic networks. In: STOC, pp. 513–522 (2010)
25. Kuhn, F., Oshman, R.: Dynamic networks: models and algorithms. *SIGACT News* 42(1), 82–96 (2011)
26. Stewart, I.A.: On the fixed-parameter tractability of parameterized model-checking problems. *Information Processing Letters* 106(1), 33–36 (2008)

# Square-Orthogonal Drawing with Few Bends per Edge

Yu-An Lin<sup>1,\*</sup> and Sheung-Hung Poon<sup>1,\*</sup>

Department of Computer Science,  
National Tsing Hua University, Hsinchu, Taiwan, R.O.C.  
`vincent9895@yahoo.com.tw, spoon@cs.nthu.edu.tw`

**Abstract.** We investigate square-orthogonal drawings of non-planar graphs with vertices represented as unit grid squares. We present quadratic-time algorithms to construct the square-orthogonal drawings of 5-graphs, 6-graphs, and 8-graphs such that each edge in the drawing contains at most two, two, and three bends, respectively. In particular, the novel analysis method we use to split a vertex so as to build some specific propagation channels in our algorithms is an interesting technique and may be of independent interest. Moreover, we show that the decision problem of determining whether an 8-graph has a square-orthogonal drawing without edge-bends is NP-complete.

## 1 Introduction

A *drawing* (or *embedding*) of a graph in the plane roughly consists of a representation of the vertices by objects, an assignment of the vertices to geometric positions in the plane, and a representation of the edges by simple open curves between the objects representing two vertices. In particular, in an *orthogonal drawing*, edges are represented by simple polygonal chains consisting of horizontal and vertical segments. Forcing a graph to be drawn orthogonally has the advantage of showing the maximal distinctiveness of adjacent edges in the drawing. The price we may need to pay for such type of clarity is to admit bends in the path representing an edge. In order to avoid confusion caused by too complex paths, it is desirable to place a low number of bends on each edge.

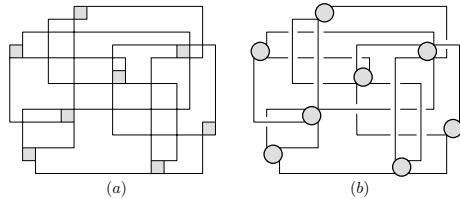
A *k-graph* is a graph of maximum vertex degree  $k$ . For an orthogonal drawing under the model that vertices are drawn as grid points (see Fig. 1(a)), Tamassia [13] showed that the planar drawing with minimum number of bends on edges for a plane 4-graph can be computed in near quadratic time. However, any plane  $k$ -graph with  $k \geq 5$  does not have orthogonal drawing. In order to draw graphs with higher vertex degree, and with simple shapes representing vertices, we investigate the orthogonal drawing with vertices represented as unit grid squares, which is called *square-orthogonal drawing* (see Fig. 1(b)). Researchers

---

\* Supported by grants 97-2221-E-007-054-MY3 and 100-2628-E-007-020-MY3 of the National Science Council (NSC), Taiwan, R.O.C.



**Fig. 1.** Vertex drawn as (i) a grid point; or (ii) a unit grid square



**Fig. 2.** (a) Vertices drawn as unit grid squares. (b) Vertices drawn as unit-diameter disks, and edges cased.

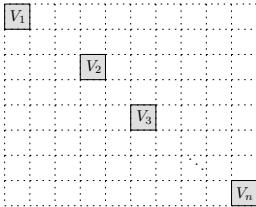
also considered even more general objects like rectangles to represent the vertices of the graph, for instance [3,4,6,7,8,11]. However, one clear disadvantage of using rectangular shapes to represent vertices is that rectangles could be fat or skinny, and hence irregular. This sometimes will hinder the readability of the whole drawing of a graph. We observe that orthogonal drawing with consistent shapes for all vertices may have the advantage of clarity for visualization. Some researchers [3,6,7] also investigated the planar orthogonal drawing under the model of representing vertices by uniformly small squares. However, in their drawings, adjacent edges are allowed to run in parallel with very small gap between them, which hinders the readability and clarity of some specific parts of the drawing. Instead, in our representation model, only edges lying on the grid lines are allowed, which prevents adjacent edges from congesting together. We focus on drawing non-planar graphs whereas some of the motivating work above only applies to planar graphs. Moreover, with our current representation model, we can afford to draw  $k$ -graphs, where  $k$  can be at most 8. Note that in our drawings, edge crossings are allowed, but edge overlapping is not allowed (see Fig. 2(a)). In our original formulation, nodes are represented as unit grid squares, and edges can cross each other; however, in practice, nodes can in fact be represented as unit-diameter disks, and the intersecting edges can be drawn using some well-known techniques, such as edge-casing (see Fig. 2(b)).

*Previous Work.* Let  $n$  and  $m$  be the number of vertices and edges in the input graph. We first consider the research work under the model of representing vertices by points on the plane. Schaffter [12] presented an algorithm to compute an orthogonal drawing, for 4-graphs, of  $2n \times 2n$  grid and with at most two bends on each edge. Biedl and Kant [2] gave a linear-time algorithm to compute an orthogonal drawing of  $n \times n$  grid area and with at most  $2n + 2$  total number of edge bends. Papakostas and Tollis [9,10] further improved the area upper bound to  $0.76n^2$  grid area while keeping the same bound on total number of edge bends. For graphs with vertex degrees higher than four, Biedl, Madden and Tollis [4] used the model of representing vertices by rectangles, and they presented an algorithm to compute an orthogonal drawing on  $(\frac{m+n}{2}) \times (\frac{m+n}{2})$  grid and with at most one bend per edge. Later, Papakosta and Tollis [11] improved the drawing area to be within  $(m-1) \times (\frac{m}{2}+2)$  while keeping the same bound on edge bends.

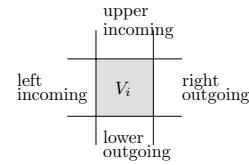
The rest of this paper is organized as follows. In Sections 2, we present some preliminaries. In Sections 3 and 4, we present algorithms to construct the square-orthogonal drawing for 5-graphs and 6-graphs such that each edge contains at most two bends, and to construct the square-orthogonal drawing of 8-graphs such that each edge contains at most three bends. In Section 5, we show that the decision problem of determining whether an 8-graph has a square-orthogonal drawing without edge-bends is NP-complete.

## 2 Preliminaries

The framework of our algorithms is as follows. At the very beginning of the algorithm, we embed the vertices  $V = \{v_1, v_2, \dots, v_n\}$  of the input graph  $G$  diagonally, beginning at the top-left corner and towards the bottom-right corner, reserving for each vertex  $v_i$  a unit square  $V_i$  (see Fig. 3). We remark that the



**Fig. 3.** The vertices are embedded along a diagonal



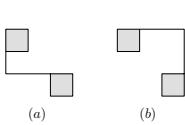
**Fig. 4.** Contact ports of a unit square node

horizontal and vertical distances between two adjacent vertices will be determined after all edges in the graph are drawn. We then proceed to insert the edges of  $G$  one by one. Each time we insert an edge, if we cannot establish an immediate connection with low bend number, we will make some modifications to the related edges so that the target bend number is attained for the inserted edge. Below we introduce some conventions and definitions.

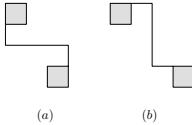
We adopt here Schaffter's [12] convention about the naming of connection contacts at the surrounding of unit squares. There can be at most eight grid edges incident to a unit square  $V_i$ , which distribute along the four sides of square  $V_i$ . A contact port is called a *left*, *right*, *upper*, or *lower* one depending on which side of  $V_i$  it locates. See Fig. 4. Suppose we try to insert edge  $v_i v_j$  by connecting unit squares  $V_i$  and  $V_j$  ( $i < j$ ). In order to obtain low bend number, it is always good to connect a right or lower contact of  $V_i$  to a left or upper contact of  $V_j$ . Thus we give these two categories of contacts different names for distinction. The left and upper contacts of  $V_i$  are called *incoming contacts* of  $V_i$ , and its right and lower contacts *outgoing contacts*. A contact is called *occupied* if it is already connected to some drawn edge; otherwise, it is called *unoccupied* or *free*.

For the convenience of our description later on, we categorize the edges according to their number of bends as follows. We call one-bend edges *L-edges* (see

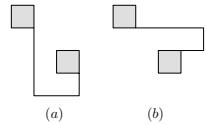
Fig. 5). Two-bend edges are classified into two types called *Z-edges* and *U-edges*, respectively (see Fig.s 6 and 7). Three-bend edges are classified into two types called *C-edges* and  $\eta$ -edges (see Fig. 8). Finally, in Fig. 9, we show two different orientations of 4-bend edges, called *S-edges*.



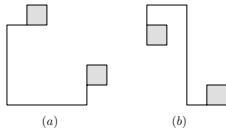
**Fig. 5.** An *L*-edge



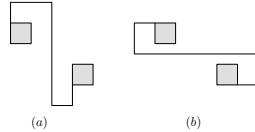
**Fig. 6.** A *Z*-edge



**Fig. 7.** A *U*-edge



**Fig. 8.** (a) A *C*-edge, and (b) an  $\eta$ -edge



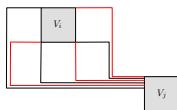
**Fig. 9.** Two 4-bend *S*-edges

### 3 Drawing 5-Graphs and 6-Graphs

In this section, we describe algorithms to draw 5-graphs and 6-graphs so that each edge contains at most two bends. We insert edges incrementally, and each time we insert an edge, we do modifications immediately so that no edges are drawn with more than two bends. As 5-graphs are subset of 6-graphs, we first present some common properties and operations for general 6-graphs. Then we proceed to consider operations specific to either kinds of graphs.

Suppose that we are currently inserting edge  $v_i v_j$  ( $i < j$ ). If either (i)  $V_j$  has a free incoming contact, or (ii)  $V_i$  has a free outgoing contact, then we will show that edge  $v_i v_j$  can be established as an edge of at most two bends. Here we only give the proof for case (i) since the proof for case (ii) is symmetric. The contact of edge  $v_i v_j$  at  $V_j$  could be left or upper contact. In any case, a connection of at most two bends can be obtained by connecting another end of edge  $v_i v_j$  to one unoccupied contact among the six left, right, or lower contacts of  $V_i$  (see Fig. 10) because the maximum degree of any vertex in  $G$  is 6.

If we do not reach any situation as in the previous paragraph, this means that we reach the situation that neither  $V_i$  contains any free outgoing contact, nor  $V_j$  contains any free incoming contact. We proceed to temporarily connect a left contact of  $V_i$  to a lower contact of  $V_j$  by a three-bend *C*-edge as shown in Fig. 11, and we justify in the following how we can modify this three-bend edge

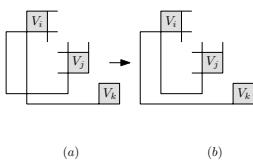


**Fig. 10.** Six contacts to connect at  $V_i$  for drawing  $v_i v_j$  with at most two bends

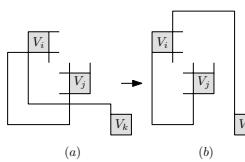


**Fig. 11.** The case that a three-bend  $C$ -edge is formed temporarily

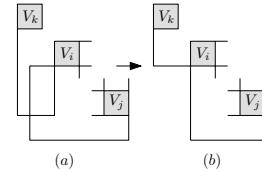
and a series of other related edges so that at the end, we result in a drawing only with edges of at most two bends. First, suppose  $V_i$  is our focus vertex for performing modifying operation. We call that the three-bend  $C$ -edge  $v_i v_j$  can be *repaired immediately* if all drawn edges becomes to possess at most two bends after some connection contacts of the edge  $v_i v_j$  and another edge are exchanged appropriately. Such a scenario is less complicated comparing to other scenarios where more modification operations are required to make the three-bend edge disappear. There are two cases for which  $C$ -edge  $v_i v_j$  can be repaired immediately: either (i) by modifying  $v_i v_j$  and another edge connecting to a lower contact of  $V_i$ , or (ii) by modifying  $v_i v_j$  and another edge connecting to a left contact of  $V_j$ . Here we only consider case (i) since the argument for case (ii) will be symmetric. We need to investigate different situations that the target edge  $v_i v_k$  connects to lower contact of  $V_i$ . If  $v_i v_k$  is an  $L$ -edge,  $C$ -edge  $v_i v_j$  can be repaired by making a switch between two contact ports of  $V_i$  from edges  $v_i v_j$  and  $v_i v_k$  (see Fig. 12). If  $v_i v_k$  is an  $Z$ -edge, the  $C$ -edge can be solved by reconnecting



**Fig. 12.**



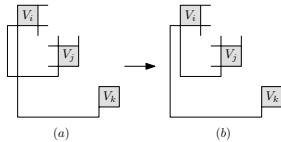
**Fig. 13.**



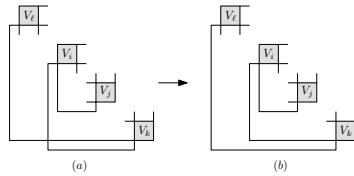
**Fig. 14.**

$V_k$  to a free upper contact of  $V_i$  (see Fig. 13). If  $v_i v_k$  is an  $U$ -edge where  $k < i$ , the  $C$ -edge can also be solved by making a switch between two contact ports of  $V_i$  from edges  $v_i v_j$  and  $v_i v_k$  (see Fig. 14). The complete argument is omitted due to lack of space.

The remaining case is that there is a lower contact of  $V_i$  that connects to  $V_k(k > i)$  with a  $U$ -edge (see Fig. 15(a)), or symmetrically there is a left contact of  $V_j$  that connects to  $V_k(k < j)$  with a  $U$ -edge. We only consider the former subcase as the latter subcase is symmetric. Now, the  $C$ -edge between  $V_i$  and  $V_j$  may not be repaired immediately by making some local modifications. Exchanging operations may only make the  $C$ -edge propagate to other edge wires. For example, the  $C$ -edge reappears as edge  $v_i v_k$  after we exchange the two contact ports at  $V_i$  from edges  $v_i v_j$  and  $v_i v_k$  (see Fig. 15). The difficulty we reach here is



**Fig. 15.** (a) A lower contact of  $V_i$  connects to  $V_k$  ( $k > i$ ) with a  $U$ -edge. (b) The  $C$ -edge reappears even after the exchanging operation.



**Fig. 16.** The  $C$ -edge  $v_iv_k$  further propagates to  $C$ -edge  $v_kv_\ell$

that the propagation of  $C$ -edges may repeat. See Fig. 16 for an example, where  $C$ -edge  $v_iv_k$  further propagates to  $C$ -edge  $v_kv_\ell$ . The bad case we worry about is that if the propagation goes into a loop, we may not be able to repair the  $C$ -edge ultimately. We have to carefully choose how to proceed for further exchanging operations. In fact, for example, if the next exchanging operation we execute is between the  $C$ -edge of  $v_iv_k$  and the  $U$ -edge  $v_iv_j$ , then we really get into an infinite loop of exchanging operations. To prevent such situation from happening, our idea is to keep switching the focus vertex where contacts are examined. For example, at the beginning of this case, we examine the lower contacts of  $V_i$  to look for candidate edges for exchanging contacts with  $C$ -edge  $v_iv_j$ ; but when  $v_iv_k$  becomes a  $C$ -edge after we exchange some contacts, we should switch our focus vertex to  $V_k$  to examine its left contacts to look for edges for possible exchanging operations with  $C$ -edge  $v_iv_k$ . For the convenience of later descriptions, we define a *leader vertex* (resp. *follower vertex*) to be the end vertex of a  $C$ -edge, whose outgoing (resp. incoming) contacts are fully occupied. For example, the above mentioned  $V_i$  and  $V_j$  are leader and follower vertices, respectively. We remark that for a leader vertex, we examine its lower contacts for further exchanging operations; but for a follower vertex, we have to examine its left contacts for further exchanging operations.

Now suppose that we result in the  $C$ -edge  $v_iv_k$  after we make the above mentioned exchanging operations. When we are examining the left contacts of  $V_k$ , we may immediately repair the  $C$ -edge in some good case as stated previously, or in the bad case, the  $C$ -edge may propagate again even after some exchanging operation. The question is whether this propagation can continue and ultimately get trapped into a loop. We report negative answer to this question in the following subsections.

### 3.1 Drawing 5-Graphs

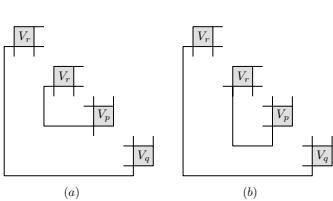
We notice that whenever we reach a  $C$ -edge, one of whose end vertices is neither a leader nor a follower, then the  $C$ -edge can be immediately repaired. Thus in order for the propagations of  $C$ -edges to continue, the end-vertices of  $C$ -edges in order of their appearances must be a series of leaders and followers. We denote this order of vertices as  $\mathcal{O} = \{V_j, V_i, V_k, V_\ell, \dots\}$  in the way that  $C$ -edge  $v_jv_i$  propagates to  $C$ -edge  $v_iv_k$ , which in turn propagates to  $C$ -edge  $v_iv_k$ , and so on.

It is clear that if two adjacent vertices in  $\mathcal{O}$  are both leaders or both followers, then the current  $C$ -edge can be immediately repaired, and the propagation ends. Hence the worst case is that the leaders and followers in the order  $\mathcal{O}$  shows up in an alternative fashion. We prove in the following lemma that the vertices in order  $\mathcal{O}$  never repeat.

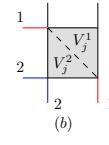
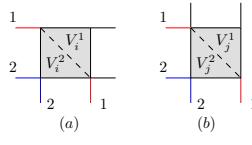
**Lemma 1.** *The vertices in the order  $\mathcal{O}$  do not repeat.*

*Proof.* Suppose to the contrary that the vertices in the order  $\mathcal{O}$  repeats, and let the first repeated vertex is vertex  $V_r$ . We further let  $V_p$  be the vertex just after the first occurrence of  $V_r$ , and  $V_q$  be the vertex just before second occurrence of  $V_r$ . Now,  $\mathcal{O}$  should appear as  $\{V_j, V_i, \dots, V_r, V_p, \dots, V_q, V_r, \dots\}$ . We have two cases depending on whether  $V_r$  is a leader or a follower. We first give the proof for former case, and then afterwards we describe the difference in the proof for the latter case.

Suppose that  $V_r$  is a leader. As  $V_p$  might be the same as  $V_q$ , we have to consider two cases, which are either  $V_p \neq V_q$  or  $V_p = V_q$ . We first consider the case that  $V_p \neq V_q$ . Suppose we are currently handling the  $C$ -edge  $v_q v_r$ . Now notice that after we handled the  $C$ -edge  $v_p v_r$  some time ago, there exists a  $U$ -edge connecting a left contact of  $V_p$  to a left contact of  $V_r$  (see Fig. 17). This is



**Fig. 17.**  $V_r$  cannot repeat



**Fig. 18.** Pairing up contacts to form two fixed channels (a) for leader vertex  $V_i$ , and (b) for follower vertex  $V_j$

**Fig. 19.** Pairing up contacts to form two fixed channels for vertex  $V_i$

the only occupied left contact of  $V_r$  since  $V_r$  is a leader. However, currently, we also have another  $C$ -edge  $v_q v_r$  connecting to an occupied left contact of  $V_r$ . This is a contradiction. Next, we consider the case that  $V_p = V_q$ . Now, the identical edges  $v_p v_r$  and  $v_q v_r$  use different ports of  $V_p$  to connect to  $V_r$  (see Fig. 17(a)). This is again a contradiction. Hence,  $V_r$  cannot repeat for this case. We now consider the latter case when  $V_r$  is a follower. Its proof is similar to that for the former case above (see Fig. 17(b)).  $\square$

This lemma implies that the  $C$ -edge can propagate at most  $n$  times across all vertices in  $G$ . Thus repairing one 3-bend  $C$ -edge takes  $O(n)$  time in the worst case. Therefore the whole graph can be drawn in  $O(n^2)$  time. We summarize our result in the following theorem.

**Theorem 1.** *There is an  $O(n^2)$ -time algorithm to compute a square-orthogonal drawing with edges of at most two bends for a 5-graph such that the drawing area is  $O(n^2)$ .*

### 3.2 Drawing 6-Graphs

In a 6-graph  $G$ , one main difference comparing to 5-graphs is that for a leader vertex  $V_i$ , there can be two edges connecting the left contacts of  $V_i$ . If any of these two edges is a  $C$ -edge, it needs to make some exchanges with some edge connecting to a lower contact of  $V_i$ . Suppose that the  $C$ -edge propagation starts with  $v_i v_j$ . Later on, it may be possible that the  $C$ -edge propagates back to become  $v_i v_k$  with connection to the other left contact of  $V_i$ . Thus the situation here is more complicated than what we reach in 5-graphs because the related vertices during the propagation of  $C$ -edges may repeat. However, we show below that the vertex  $V_i$  cannot be visited for the third time during the propagation. Since the two edges connecting to the two left contacts of  $V_i$  may make some exchanges with the two edges connecting to the two lower contacts of  $V_i$ , for the convenience of our analysis, we pair up the contact ports to form two fixed channels as shown in Fig. 18(a). (The paired contacts to form channels for follower vertex  $V_j$  is shown in Fig. 18(b).) According to the two fixed channels, we can treat the single vertex  $V_i$  as being split into two small vertices, say  $V_i^1$  and  $V_i^2$ . Thus when  $V_i$  does repeat for the first time in the propagation sequence of the  $C$ -edges, we can treat this situation as if we reach a new vertex by using the above split vertices. Hence, the similar arguments as in Lemma 1 can be used to argue that there will be no repeats on the sequence of split vertices along the propagation of  $C$ -edges. So we obtain the following lemma.

**Lemma 2.** *Any vertex appeared during the propagation of  $C$ -edges cannot repeat for the third time.*

We use similar argument as for Theorem 1 to obtain the following theorem.

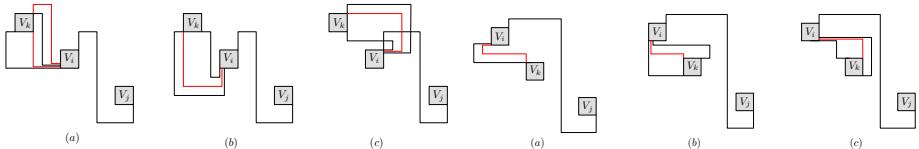
**Theorem 2.** *There is an  $O(n^2)$ -time algorithm to compute a square-orthogonal drawing with edges of at most two bends for a 6-graph such that the drawing area is  $O(n^2)$ .*

## 4 Drawing 8-Graphs

In this section, we describe the algorithm to draw 8-graphs so that each edge contains at most three bends. Again, we insert edges incrementally, and each time we insert an edge, we do modifications immediately so that no edges are drawn with more than three bends.

Suppose that we are currently inserting edge  $v_i v_j (i < j)$ . Since both unit squares  $V_i$  and  $V_j$  have four sides,  $v_i v_j$  has 16 combination ways to do connection depending on the availability of the contact ports of  $V_i$  and  $V_j$ . Among all these combinations, there are only two ways a 4-bend  $S$ -edge can be possibly created:

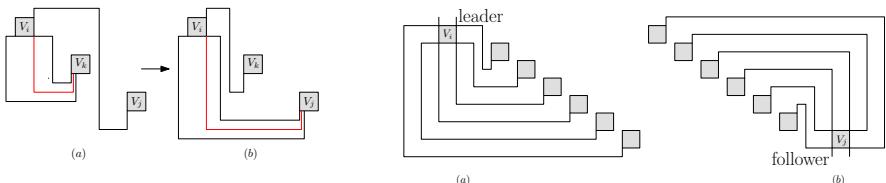
(i) an upper contact of  $V_i$  connects to a lower contact of  $V_j$  (see Fig. 9(a)); and (ii) a left contact of  $V_i$  connects to a right contact of  $V_j$  (see Fig. 9(b)). If any of these two cases happens, we need to proceed to perform some modifying operations to repair the 4-bend edge  $v_i v_j$ . As handling case (ii) is symmetric to case (i), we investigate here the details for case (i) only. Suppose  $V_i$  is our current focus vertex for performing modifying operation. First, we examine the lower contacts of  $V_i$  to look for opportunities for modifying operations. If there is still a free lower contact at  $V_i$ , then  $S$ -edge  $v_i v_j$  can be repaired immediately by changing the connection contact of edge  $v_i v_j$  to the free lower contact. Otherwise, both lower contacts of  $V_i$  are occupied. We will then try to perform exchanging operations on edge  $v_i v_j$  and another edge  $v_i v_k$  of at most 3 bends of  $V_i$ . The edge  $v_i v_k$  may link to a left contact of  $V_i$  (see Fig. 20(a) and Fig. 21(a)), a lower contact of  $V_i$  (see Fig. 20(b) and Fig. 21(b)), or a right contact of  $V_i$  (see Fig. 20(c) and Fig. 21(c)). These cases shown in these figures are the cases that  $S$ -edge  $v_i v_j$  can be repaired immediately by local modifying operations. The detailed arguments are omitted due to lack of space. Apart from these



**Fig. 20.** The cases that  $S$ -edge  $v_i v_j$  can be repaired immediately when  $k < i$

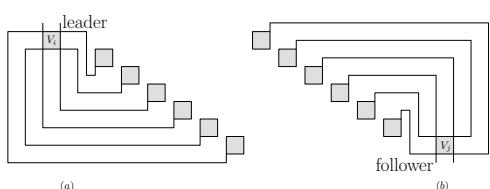
**Fig. 21.** The cases that  $S$ -edge  $v_i v_j$  can be repaired immediately when  $k > i$

easy cases as shown in Figs 20 and 21, we still have three remained cases (see Fig. 22(a)): (i)  $v_i v_k(k > i)$  is a  $C$ -edge connecting a left contact of  $V_i$  and a lower contact of  $V_k$ ; (ii)  $v_i v_k(k > i)$  is a  $U$ -edge connecting a lower contact of  $V_i$  and a lower contact of  $V_k$ ; and (iii)  $v_i v_k(k > i)$  is a  $\eta$ -edge connecting a right contact of  $V_i$  and a lower contact of  $V_k$ . For such cases,  $S$ -edge  $v_i v_j$  may not be repaired immediately by making some local exchanging operations. Local exchanging operations may only make  $S$ -edge  $v_i v_j$  propagate to some other edge wire. For example, in Fig. 22(b), the  $S$ -edge reappears as edge  $v_i v_k$  after we exchange the contacts at  $V_i$  of edges  $v_i v_j$  and  $v_i v_k$ . We will show that



**Fig. 22.**  $S$ -edge  $v_i v_j$  propagates to another  $S$ -edge  $v_i v_k$

**Fig. 23.** (a) A leader, and (b) a follower



the propagation sequence of  $S$ -edges does not get trapped into a loop. For this purpose, we use the similar idea as in Section 3 by keeping switching the current focus vertex for operations whenever some modifying operation is performed. We take Fig. 22 as an example to illustrate our propagation procedure. First, we examine the lower contacts of  $V_i$  to look for candidate edges for exchanging contacts; but when  $v_i v_k$  becomes an  $S$ -edge after we exchange some contacts, we switch our focus vertex to  $V_k$  to examine its upper contacts to look for edges for further modifying operations, and so forth. For our convenience, we define a *leader vertex* to be the upper end vertex  $V_i$  of an  $S$ -edge, whose six left, right, and lower contacts connect to some lower contacts of some other vertices below  $V_i$ , respectively. A *follower vertex* is defined to be the lower end vertex  $V_j$  of an  $S$ -edge, whose six left, right, and upper contacts connect to some lower contacts of some other vertices above  $V_j$ , respectively. See Fig. 23 for an example. We note here that the definitions of leader and follower vertices are different from those in Section 3.

Now, as in Section 3.2, we split each leader or follower vertex  $V_i$  to become two small vertices,  $V_i^1$  and  $V_i^2$ , as shown in Fig. 19; and we establish two fixed channels for the propagation of  $S$ -edges to go through  $V_i$ . Then using similar arguments as for Lemma 2, we can obtain the following lemma, whose complete proof is omitted.

**Lemma 3.** *Any vertex appeared during the propagation of  $S$ -edges cannot repeat for the third time.*

Hence we have the following theorem.

**Theorem 3.** *There is an  $O(n^2)$ -time algorithm to compute a square-orthogonal drawing with edges of at most three bends for a 8-graph such that the drawing area is  $O(n^2)$ .*

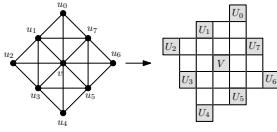
## 5 Hardness Result

In this section, we show the NP-hardness of deciding whether there is a square-orthogonal drawing without bends on edges under our vertex representation model for an 8-graph. We provide a sketchy proof here, and the complete proof is omitted due to lack of space.

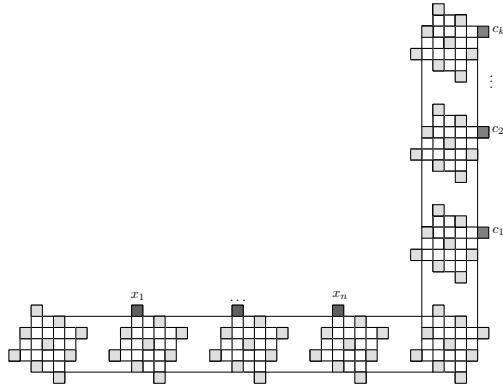
**Theorem 4.** *The problem of deciding whether an 8-graph has a square-orthogonal drawing without bends on edges is NP-complete.*

*Proof. (Sketch.)* The problem is clearly in NP. To prove the NP-hardness of our problem, we reduce the 3SAT problem to our problem. Let  $\{x_1, x_2, \dots, x_n\}$  be a set of  $n$  variables, and let  $\{c_1, c_2, \dots, c_k\}$  be a set of  $k$  clauses, each of which contains exactly 3 literals. The 3SAT problem is to decide whether there is a truth assignment to the  $n$  variables so that all the  $k$  clauses are true.

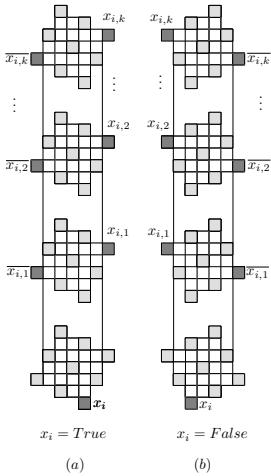
In our polynomial-time reduction, we construct the basic construction unit called a *diamond*, and three important gadgets called *skeleton*, *variable towers*,



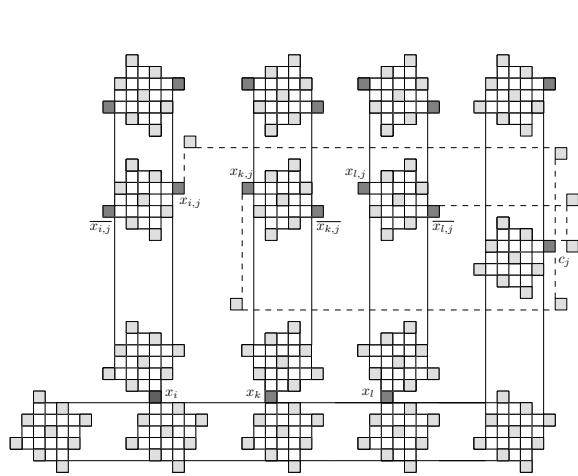
**Fig. 24.** The diamond structure



**Fig. 25.** Skeleton containing vertex ports for linking to variable towers and clause linkages



**Fig. 26.** Variable tower for  $x_i$  and its truth-value representations



**Fig. 27.** Clause gadget for clause  $c_j = x_i \vee x_k \vee \bar{x}_l$

and *clause linkages*. The diamond structure is shown in Fig. 24, which can be proven to be *stable* in the sense the relative positions of the eight surrounding vertex squares relative to the center vertex square are fixed. (Its proof is omitted here.) Using this structure as basic construction units, we construct the skeleton as shown in Fig. 25, where the upper spikes along the horizontal list of diamonds will connect to the variable towers, which we will construct later on, and the right spikes along the vertical list of diamonds will connect to the clause linkages, which we will construct later on.

We construct the variable tower for variable  $x_i$  as shown in Fig. 26(a). Each of the two possible embeddings of the variable tower represents the *True* and

*False* values for  $x_i$ , respectively. See Fig. 26(a) and (b). Note that the literals on the right hand side of the tower always has the *True* value in either way the tower is embedded.

Next, we describe the construction of the clause linkage for clause  $c_j$ . As an example, we suppose that  $c_j = x_i \vee x_k \vee \overline{x_l}$ . We connect vertex  $c_j$  on the skeleton to literals  $x_{i,j}, x_{k,j}, \overline{x_{l,j}}$ , respectively, with a 3-path. See Fig. 27. We observe that  $c_j = \text{True}$  corresponds to the 3-edge path connecting to a right port of node  $c_j$ . With this observation, it is not hard to show that the given 3SAT formula is satisfiable iff the constructed graph has a square-orthogonal drawing without bends on edges. The complete proof is omitted due to lack of space.  $\square$

## References

1. Bertolazzi, P., Battista, G.D., Didimo, W.: Computing orthogonal drawings with minimum number of bends. *IEEE Trans. on Computers* 49(8), 826–840 (2000)
2. Biedl, T.C., Kant, G.: A better heuristic for orthogonal graph drawings. *Computational Geometry: Theory and Applications* 9, 159–180 (1998)
3. Biedl, T.C., Kaufmann, M.: Area-efficient static and incremental graph drawings. In: Burkard, R.E., Woeginger, G.J. (eds.) *ESA 1997*. LNCS, vol. 1284, pp. 37–52. Springer, Heidelberg (1997)
4. Biedl, T.C., Madden, B.P., Tollis, I.G.: Drawing high-degree graphs with small grid-size. Technical Report 37–96, RUTCOR, Rutgers University (November 1996)
5. Formann, M., Hagerup, T., Haralambides, J., Kaufmann, M., Leighton, F.T., Symvonis, A., Welzl, E., Woeginger, G.J.: Drawing graphs in the plane with high resolution. In: *Proceedings IEEE Symposium on FOCS*, pp. 86–95 (1990); *SIAM Journal on Computing* 22(5), 1035–1052 (1993)
6. Fößmeier, U., Kaufmann, M.: Algorithms and Area Bounds for Nonplanar Orthogonal Drawings. In: DiBattista, G. (ed.) *GD 1997*. LNCS, vol. 1353, pp. 134–145. Springer, Heidelberg (1997)
7. Fößmeier, U., Kaufmann, M.: Drawing high degree graphs with low bend numbers. In: Brandenburg, F.J. (ed.) *GD 1995*. LNCS, vol. 1027, pp. 254–266. Springer, Heidelberg (1996)
8. He, X.: A simple linear time algorithm for proper box rectangular drawings of plane graphs. *Journal of Algorithms* 40(1), 82–101 (2001)
9. Papakostas, A., Tollis, I.G.: A pairing technique for area-efficient orthogonal drawings. In: North, S.C. (ed.) *GD 1996*. LNCS, vol. 1190, pp. 355–370. Springer, Heidelberg (1997)
10. Papakostas, A., Tollis, I.G.: Algorithms for area-efficient orthogonal drawings. *Computational Geometry: Theory and Applications* 9(1–2), 83–110 (1998)
11. Papakostas, A., Tollis, I.G.: Efficient orthogonal drawings of high degree graphs. *Algorithmica* 26(1), 100–125 (2000)
12. Schaffter, M.: Drawing graphs on rectangular grids. *Discrete Applied Math.* 63(1), 75–89 (1995)
13. Tamassia, R.: On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing* 16(3), 421–444 (1987)
14. Tamassia, R.: Planar orthogonal drawings of graphs. In: Proc. of the IEEE International Symposium on Circuits and Systems, pp. 319–322 (1990)
15. Valiant, L.: Universality considerations in VLSI circuits. *IEEE Trans. on Computers* 30(2), 135–140 (1981)

# Covering Tree with Stars

Jan Baumbach<sup>1,3,\*</sup>, Jiong Guo<sup>2,\*</sup>, and Rashid Ibragimov<sup>1,\*\*</sup>

<sup>1</sup> Max Planck Institute für Informatik, Saarbrücken 66123, Germany  
`ribragim@mpi-inf.mpg.de`

<sup>2</sup> Universität des Saarlandes, Campus E 1.7, Saarbrücken 66123, Germany  
`jguo@mmci.uni-saarland.de`

<sup>3</sup> University of Southern Denmark, Campusvej 5, 5230 Odense M, Denmark  
`jan.baumbach@imada.sdu.dk`

**Abstract.** We study the tree edit distance problem with edge deletions and edge insertions as edit operations. We reformulate a special case of this problem as COVERING TREE WITH STARS (CTS): given a tree  $T$  and a set  $\mathcal{S}$  of stars, can we connect the stars in  $\mathcal{S}$  by adding edges between them such that the resulting tree is isomorphic to  $T$ ? We prove that in the general setting, CST is NP-complete, which implies that the tree edit distance considered here is also NP-hard, even when both input trees having diameters bounded by 10. We also show that, when the number of distinct stars is bounded by a constant  $k$ , CTS can be solved in polynomial time by presenting a dynamic programming algorithm running in  $O(|V(T)|^2 \cdot k \cdot |V(\mathcal{S})|^{2k})$  time.

**Keywords:** graph algorithms, tree edit distance, NP-completeness, dynamic programming.

## 1 Introduction

Given two graphs  $G_1$  and  $G_2$ , the *edit distance* between  $G_1$  and  $G_2$  is defined to be the minimum number of edit operations to transform  $G_1$  to  $G_2$ . Graph edit distance is one of the major measures for studying the similarity of graphs and has found applications in many areas [1,2,3]. However, in the general problem statement application of the edit distance is limited by its complexity, since computing edit distance between two graphs is not easier than the subgraph isomorphism problem, which is NP-complete [4].

Restricting graphs to trees branched in investigating the tree edit distance. Hereby, one distinguishes mostly ordered and unordered versions, for which such edit operations as vertex deletion, vertex insertion, and vertex relabeling are applied. With this set of the edit operations, for which the hierarchy as well as vertex labels are important, the edit distance between trees and forests has been extensively studied [5,6,7]. For the ordered version of tree edit distance, an algorithm requiring  $O(n^3)$  time was developed [8], where  $n$  denotes the size

---

\* Partially supported by the DFG Excellence Cluster MMCI.

\*\* Partially supported by the International Max Planck Research School.

of the input trees or forests. The tree edit distance in the unordered case was shown to be NP-hard [9] and MAX SNP-hard [10]. In [5], a fixed-parameter algorithm running in  $O(2.62^k \cdot \text{poly}(n))$  time is presented for unordered version, where  $k$  is the upper bound of the edit distance. A short review and further recent developments of the problem can be found in [11].

In this paper we aim at exploring the complexity of computing the edit distance between trees with the edit operations affecting only the edges, namely, deleting and inserting edges:

#### TREE EDIT DISTANCE WITH EDGE INSERTION AND DELETION (TED-ID)

**Input:** Trees  $T_1$  and  $T_2$ , a non-negative integer  $d$

**Question:** Can we modify  $T_1$  by adding or deleting at most  $d$  edges in it such that the resulting tree is isomorphic to  $T_2$ ?

TED-ID is partially motivated by alignment of a backbone trees extracted from biological networks of different species in Bioinformatics. Here, large continuous regions of aligned subtrees can serve as strong indication for biologically meaningful node-to-node correspondence, easing at the same time matching of the other not matched nodes of the networks (similar to [12]). Note that the problem of modifying a given graph by deleting and inserting edges, such that the resulting graph meets some specified properties, has been extensively studied in the last years, motivated by various applications from Bioinformatics, data mining, etc. For most classical graph properties, the corresponding modification problems turned out to be NP-hard [13,14].

Given a solution for TED-ID, we can perform the edit operations in the order of first deleting edges from the first tree, resulting in a forest, and then inserting other edges to connect the forest. Thus, the problem of transforming a given forest to a given tree by deleting zero edges and adding a number of edges can be considered as a separate phase of the tree edit distance problem. We observe that, the NP-completeness of the subforest isomorphism problem [4] implies that this edge addition problem is NP-complete. By a simple reduction from 3-PARTITION [4], we can even prove the NP-hardness for the forest being a collection of paths.

Motivated by these NP-hardness results, we consider another restriction on the components of the forest, that is, all components being stars, and intend to find a boundary of the complexity for the tree edit distance problem:

**Input:** A collection of stars  $\mathcal{S}$  and a tree  $T$ .

**Question:** Can we connect the stars in  $\mathcal{S}$  by adding edges between them such that the resulting tree is isomorphic to  $T$ ?

We call this problem COVERING TREE WITH STARS (CTS), since one can consider CTS as using the stars to covering the vertices of the tree. We use  $|\mathcal{S}|$  to denote the number of stars in  $\mathcal{S}$  and  $k$  the number of distinct stars in  $\mathcal{S}$ . Moreover, let  $V(\mathcal{S})$  and  $V(T)$  be the set of the vertices in the stars of  $\mathcal{S}$  and  $T$ , respectively. Clearly, we can assume  $|V(\mathcal{S})| = |V(T)|$ . Moreover, the number of edges added to the stars is exactly  $|\mathcal{S}| - 1$ . CTS can also be formulated as

a matching problem between  $\mathcal{S}$  and  $T$ . Here, one is seeking for a one-to-one mapping  $f$  from  $V(\mathcal{S})$  to  $V(T)$  such that  $f$  “preserves” the edges in  $\mathcal{S}$ , that is, if  $u, v \in V(\mathcal{S})$  are adjacent, then  $f(u)$  and  $f(v)$  must be adjacent. We obtain a classification of the complexity of CTS with respect to the number  $k$  of distinct stars in  $\mathcal{S}$ : CTS can be solved in polynomial time, if  $k$  is bounded by a constant; otherwise, this problem is NP-complete. The latter is shown again by a reduction from 3-PARTITION. As a corollary of this NP-complete result, we show that TED-ID remains NP-hard, even when both trees have bounded diameters.

*Preliminaries.* Throughout the paper, we consider only simple, undirected graphs without self-loop. Given a graph  $G$ , we use  $V(G)$  to denote the vertex set of  $G$ . The neighborhood of a vertex in a graph  $G$ , denoted as  $N(v)$ , is the set of vertices which are adjacent to  $v$ . The degree of  $v$  is then the size of  $N(v)$ . A forest is an graph without cycle, while a tree is a connected forest. The degree-one vertices of a tree are called the leaves and the others are the internal vertices. A star is a tree with at most one internal vertex. This internal vertex is then called the center of this star. The size of a star  $S$  is the number of edges in  $S$ . The length of a path is the number of edges in this path.

## 2 NP-Completeness Results

We first show that the edge addition problem to transform a forest to a tree is NP-hard even for a forest being a collection of paths. Then we show the NP-hardness of the general setting of COVERING TREE WITH STARS (CTS), that is, there are unbounded many distinct stars in  $\mathcal{S}$ . As a corollary of this hardness, we prove that TED-ID is NP-hard, even when both trees have bounded diameters.

**Theorem 1.** *Given a collection of paths  $F$  and a tree  $T$ , transforming  $F$  to  $T$  by edge additions is NP-hard.*

*Proof.* To prove the NP-hardness, we reduce from the 3-PARTITION problem defined as follows.

**Input:** A multiset  $X$  of  $3m$  integers  $x_1, \dots, x_{3m}$  and an integer  $B$ .

**Question:** Can  $X$  be partitioned into  $m$  subsets such that each subset contains exactly three integers and the sum of the integers in each subset is  $B$ ?

Note that that 3-PARTITION remains NP-hard, even when every integer in  $X$  is bounded by a polynomial in  $m$  [4].

Given an instance  $(X, B)$ , then the corresponding collection  $F$  consists of  $3m$  paths of length  $x_i - 1$  for  $i = 1, \dots, 3m$ . Every path corresponds to one distinct element of set  $X$ . There is also one additional path  $p$  of length  $2B$ . The corresponding tree  $T$  is composed of  $m + 2$  paths of length  $B - 1$  connected to one additional vertex  $r$  with  $m + 2$  edges; for each of these paths, there is an edge between  $r$  and one of its end-vertices.

To connect all  $3m + 1$  paths from  $F$  by  $3m$  edge additions, we are forced to map the middle vertex of path  $p$  to vertex  $r$  in the tree. The other vertices of  $p$  have to be mapped to two paths of length  $B - 1$  in  $T$ . It is not hard to see that the rest  $3m$  paths from  $F$  can be mapped to unmapped vertices in  $T$ , if and only if there is a 3-partition of set  $X$ .  $\square$

**Theorem 2.** *CTS is NP-complete.*

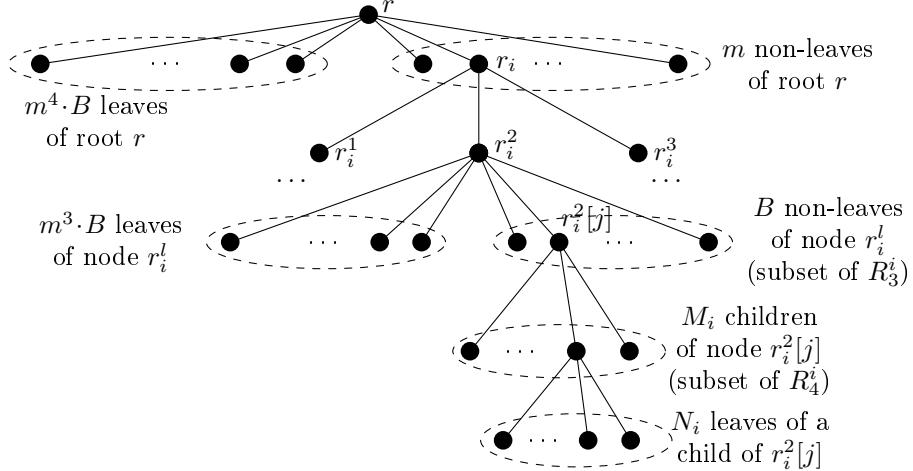
*Proof.* Clearly, CTS is in NP, since we can guess  $|\mathcal{S}| - 1$  edges between the stars in  $\mathcal{S}$  in polynomial time and the isomorphism testing between two trees can be done in polynomial time [15]. We reduce again from 3-PARTITION.

Given an instance  $(X, B)$ , we can safely assume that  $0 \leq x_i < B$  for each  $i = 1, \dots, 3m$ . We construct an instance  $(\mathcal{S}, T)$  of CTS in the following way. First,  $\mathcal{S}$  consists of five subcollections of stars. The first one has only one “large” star, denoted as  $S_L$ , which has  $m^4 \cdot B + m$  leaves. The second subcollection  $\mathcal{S}_2$  contains  $3m$  stars, one-to-one corresponding to the integers in  $X$ . For each integer  $x_i$  with  $i = 1, \dots, 3m$ , we add a star with  $m^3 \cdot B + x_i$  many leaves to  $\mathcal{S}_2$ . The third subcollection  $\mathcal{S}_3$  contains  $2B \cdot m$  stars. More precisely, for each  $i = 1, \dots, m$ , we create a collection  $\mathcal{S}_3^i$ , which contains  $2B$  stars, each having  $M_i := m^2 \cdot B + i$  many leaves. We set  $\mathcal{S}_3 := \bigcup_{i=1}^m \mathcal{S}_3^i$ . Moreover, we create for each  $i = 1, \dots, m$  a collection  $\mathcal{S}_4^i$  with  $B \cdot M_i$  stars, each having  $N_i := m \cdot B + i$  leaves, and set  $\mathcal{S}_4 := \bigcup_{i=1}^m \mathcal{S}_4^i$ . Finally, the last subcollection  $\mathcal{S}_5$  consists of  $\sum_{i=1}^m (m \cdot 2B \cdot M_i \cdot N_i)$  many “singletons”. A singleton is an isolated vertex.

Next, we construct the tree  $T$ . To ease the description, we describe  $T$  as a tree rooted at a vertex  $r$ , which has  $m^4 \cdot B$  many leaves as children (see also Fig. 1 for the depiction of tree  $T$ ). In addition,  $r$  has  $m$  children  $r_1, \dots, r_m$ , which are the roots of  $m$  subtrees, denoted as  $T_1, \dots, T_m$ . Let  $R_1$  be the set of  $r$ 's children. For each  $i = 1, \dots, m$ , the vertex  $r_i$  has three children  $r_i^l$  with  $l = 1, 2, 3$ . Let  $R_2$  be set containing all  $3m$  children of the non-leaf vertices in  $R_1$ . Each vertex  $r_i^l \in R_2$  has  $m^3 \cdot B$  leaves as children. In addition,  $r_i^l$  has other  $B$  children  $r_i^l[j]$  with  $j = 1, \dots, B$ , at which the subtrees  $T_i^l[j]$  are rooted. Let  $R_3^i$  be the set of the non-leaf children  $r_i^l[j]$  of  $r_i^l$  for all  $l = 1, 2, 3$ . Clearly,  $|R_3^i| = 3B$ . We set  $R_3 := \bigcup_{i=1}^m R_3^i$ . Finally, in each  $T_i^l[j]$ ,  $r_i^l[j]$  has  $M_i = m^2 \cdot B + i$  many children, each of which has again  $N_i = m \cdot B + i$  many leaves as children. The set  $R_4^i$  contains all children of the vertices in  $R_3^i$ , that is,  $|R_4^i| = 3B \cdot M_i$ . We set  $R_4 := \bigcup_{i=1}^m R_4^i$ . This completes the construction of  $T$ .

Since every integer and thus  $B$  are bounded by a polynomial of  $m$ , the construction of  $(\mathcal{S}, T)$  clearly needs polynomial time. Next, we prove now that  $(X, B)$  is a yes-instance iff  $(\mathcal{S}, T)$  is a yes-instance.

“ $\implies$ ”: Let  $X_1, \dots, X_m$  be a partition of  $X$  with  $|X_i| = 3$  and  $\sum_{x \in X_i} x = B$  for each  $i = 1, \dots, m$ . A mapping  $f$  between  $V(\mathcal{S})$  and  $V(T)$  can be derived as follows. First, map the center of  $S_L$  to the root  $r$  of  $T$  and  $S_L$ 's leaves to the children of  $r$ . Then, for each  $i = 1, \dots, m$ , let  $X_i = \{x_i^1, x_i^2, x_i^3\}$ . For each  $l = 1, 2, 3$ , we map the center of the star  $S$  in  $\mathcal{S}_2$ , which corresponds to  $x_i^l$ , to the child  $r_i^l$  of  $r_i$ . Note that  $r_i$  is a child of  $r$  and the root of the subtree  $T_i$ . Moreover,  $m^3 \cdot B$  many leaves of  $S$  are mapped to the  $m^3 \cdot B$  leaf children of  $r_i^l$ . The remaining  $x_i^l$  leaves



**Fig. 1.** The tree  $T$  of CTS corresponding to an instance of 3-PARTITION

of  $S$  are then mapped to the non-leaf children of  $r_i^l$ . Note that  $r_i^l$  has now  $B - x_i^l$  unmapped children and  $r_i^1, r_i^2, r_i^3$  have together  $2B$  unmapped children. Let  $D$  be the set of unmapped children of  $r_i^1, r_i^2, r_i^3$  and  $E := R_3^i \setminus D$ . Clearly,  $D \subseteq R_3^i$ . Now, we map the centers of the stars in  $\mathcal{S}_3^i$  to the vertices in  $D$ . Since each of the vertices in  $D$  has exactly  $M_i = m^2 \cdot B + i$  many children, the leaves of the stars in  $\mathcal{S}_3^i$  can be mapped to the children of the vertices in  $D$ . By this way, all stars in  $\mathcal{S}_3$  can be mapped to the tree. Note that each of the children of the vertices in  $D$  has  $N_i = m \cdot B + i$  leaves as children, which are still unmapped. Let us now consider the stars in  $\mathcal{S}_4$ . Recall that  $|\mathcal{S}_4^i| = B \cdot M_i$  and each star in  $\mathcal{S}_4^i$  has  $N_i$  leaves. Note that the vertices in  $E$  are mapped to the leaves of the stars in  $\mathcal{S}_2^i$  and each vertex in  $E$  has  $M_i$  many children, each of which has again  $m \cdot B + i$  leaves as children. By  $|E| = B$ , we can conclude that the stars in  $\mathcal{S}_4^i$  can be mapped to the subtrees rooted at the children of the vertices in  $E$ . Finally, recall that, in the above analysis, each of the children of the vertices in  $D$  has  $N_i := m \cdot B + i$  unmapped leaves as children. Thus, in each subtree  $T_i$  there are exactly  $2B \cdot M_i \cdot N_i$  many unmapped leaves. Summing up over all  $T_i$ 's, the singletons in  $\mathcal{S}_5$  can then be mapped. Since we always map the leaves of a star together with its center, the mapping clearly satisfies the edge-preserving condition.

“ $\Leftarrow$ ”: Let  $f$  be a mapping from  $V(\mathcal{S})$  to  $V(T)$ . In order to prove this direction, we need the following claims. The first three claims follow directly from the degrees of the root and the vertices in  $R_2$  and  $R_3$ .

**Claim 1.** The center of the large star  $S_L$  has to be mapped to the root  $r$  of  $T$ , and the leaves of  $S_L$  one-to-one to the vertices in  $R_1$ .

**Claim 2.** The centers of the stars in  $\mathcal{S}_2$  have to be mapped one-to-one to the vertices in  $R_2$ .

**Claim 3.** The centers of the stars in  $\mathcal{S}_3$  have to be mapped to the vertices in  $R_3$ .

By Claim 2, the stars in  $\mathcal{S}_2$  and thus the integers in  $X$  are partitioned by the mapping  $f$  into  $m$  subsets, denoted by  $X_1, \dots, X_m$ , such that  $|X_i| = 3$  for all  $i = 1, \dots, m$ . It remains to prove that  $\sum_{x \in X_i} x = B$  for all  $i = 1, \dots, m$ . To this end, we need the following claim, which is true, since a star in  $\mathcal{S}_2$  has more leaves than the number of leaf children of a vertex in  $R_2$  and mapping a leaf in  $T$  to a singleton is never better than mapping it to a leaf of star in  $\mathcal{S}$ .

**Claim 4.** If  $(\mathcal{S}, T)$  is a yes-instance, then there is a mapping such that the leaf children of the vertices in  $R_2$  are all mapped to the leaves of the stars in  $\mathcal{S}_2$ .

According to Claims 3 and 4,  $2mB$  vertices in  $R_3$  have to be mapped to the centers of the stars in  $\mathcal{S}_3$  and the remaining vertices have to be mapped to the leaves of the stars in  $\mathcal{S}_2$ . Since we can w.l.o.g. assume  $\sum_{x \in X} x = mB$ , the following claim holds.

**Claim 5.** The centers of the stars in  $\mathcal{S}_4$  have to be mapped to the vertices in  $R_4$ .

In order to prove  $\sum_{x \in X_i} x = B$  for all  $i = 1, \dots, m$ , we apply an induction from  $i = m$  to  $i = 1$ . For  $i = m$ , let  $X_m = \{x_m^1, x_m^2, x_m^3\}$  and let  $S_1, S_2, S_3$  denote the three stars in  $\mathcal{S}_2$ , which correspond to the integers in  $X_m$  and whose roots are mapped to  $r_m^1, r_m^2, r_m^3$ , respectively. By Claim 4, totally  $\sum_{j=1}^3 x_m^j$  leaves of the stars  $S_1, S_2, S_3$  have to be mapped to the vertices in  $R_3^m$ . Moreover, Claim 3 and the fact that the stars in  $\mathcal{S}_3^m$  have more leaves than all other stars in  $\mathcal{S}_3^j$  with  $j < m$  imply that the centers of the stars in  $\mathcal{S}_3^m$  have to be mapped to the vertices in  $R_3^m$ . With  $|\mathcal{S}_3^m| = 2B$  and  $|R_3^m| = 3B$ , we know  $\sum_{j=1}^3 x_m^j \leq B$ . Let  $D$  be the set of vertices in  $R_3^m$  which are mapped to the leaves of  $S_1, S_2, S_3$ ,  $E_1$  be the set of vertices in  $R_3^m$  that are mapped to the centers of the stars in  $\mathcal{S}_3^m$ , and  $E_2 := R_3^m \setminus (D \cup E_1)$ . Clearly,  $|D| = \sum_{j=1}^3 x_m^j$  and  $|E_2| = B - |D|$ . If  $E_2 \neq \emptyset$ , then the vertices from  $E_2$  can be mapped to the centers of other stars. However, by Claims 3 and 4, all  $E_2$ -vertices have to be mapped to the centers of some stars from  $\bigcup_{j=1}^{m-1} \mathcal{S}_3^j$ , since  $\sum_{x \in X} x = mB$  implies that  $\sum_{j=1}^3 x_i^j > B$  for some  $i < m$ . Now, consider the stars in  $\mathcal{S}_4^m$ . Since all vertices in  $S_1 \cup S_2 \cup S_3$  are mapped and the stars in  $\mathcal{S}_4^m$  have more leaves than the degrees of the vertices in  $R_4^j$  with  $j < m$ . The centers of these stars can only be mapped to the vertices in  $R_4^m$ . However, by mapping the centers of the stars in  $\mathcal{S}_3^m$  to the vertices in  $E_1$  and the centers of some stars from  $\mathcal{S}_3^i$  for some  $i < m$  to the vertices in  $E_2$ , some of the vertices in  $R_4^m$  are already mapped. For the vertices in  $E_1$ , all their children are mapped. Moreover, since  $|D| + |E_2| = B$  and at least one of the children of each of the  $E_2$ -vertices is mapped, we have at most  $|D| \cdot M_m + |E_2| \cdot (M_m - 1)$  vertices in  $R_4^m$  not mapped to the stars in  $\mathcal{S}_3^m$ . Recall that every vertex in  $R_3^m$  has  $M_m = m^2 \cdot B + m$  many children. However, by  $|\mathcal{S}_4^m| = B \cdot M_m$ , the centers of some stars in  $\mathcal{S}_4^m$  cannot be mapped to the vertices in  $R_4^m$ , a contradiction. Thus, we have  $B = |D|$  and  $\sum_{j=1}^3 x_m^j = B$ . This means that  $E_2 = \emptyset$  and all vertices in  $R_3^m \cup R_4^m$  are mapped to stars in  $\mathcal{S}_3^m \cup \mathcal{S}_4^m$ . Thus, the induction step from  $i+1$  to  $i$  can be conducted in a similar way as for the case  $i = m$ . The reason for this

is that the center of all stars in  $\mathcal{S}_3^i$  can only be mapped to the vertices in  $R_3^i$ . This holds also for the stars in  $\mathcal{S}_4^i$  and the vertices in  $R_4^i$ . In summary, we can derive a partition from the mapping  $f$  such that every subset  $X_i$  has exactly three integers, and  $\sum_{x \in X_i} x = B$  for each  $i = 1, \dots, m$ .  $\square$

**Corollary 1.** *TED-ID is NP-hard even when the diameters of both trees are bounded by 10.*

*Proof.* We construct an equivalent instance of TED-ID from the CTS-instance constructed in the proof of Theorem 2.

Let  $x$  be the number of the stars in the CTS-instance  $(\mathcal{S}, T)$  in the proof of Theorem 2. We construct tree  $T_1$  as follows. First, create  $x$  “big stars”, each with  $m^8 \cdot B^3$  leaves. Then, connect the stars in  $\mathcal{S}$  one-to-one to the big stars: for each star in  $\mathcal{S}$ , add an edge between its center and one of the leaves of the corresponding big star. Finally, create a star with  $m^4$  leaves, and add an edge between the center of this star and each of the centers of the big stars. Clearly, the resulting tree  $T_1$  has a diameter equal to 8.

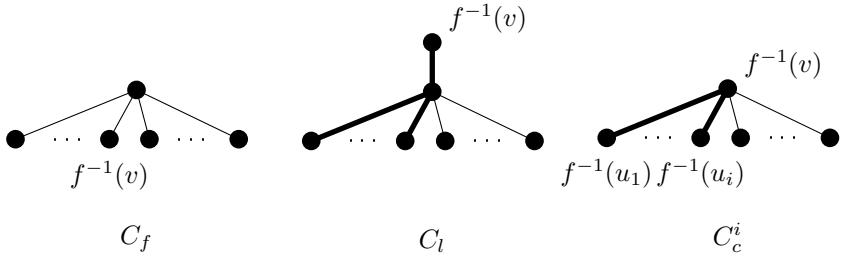
The second tree  $T_2$  is firstly set equal to the tree  $T$  of the CTS-instance from the proof of Theorem 2. Then, add  $x$  “big” stars, each having  $m^8 \cdot B^3$  leaves. Finally, add an edge between the root  $r$  of  $T$  and each of the centers of the big stars. The diameter of the resulting tree  $T_2$  is clearly equal to 10.

We set  $d := 2 \cdot (x - 1)$ . From the construction of  $(\mathcal{S}, T)$ , we have  $d = O(m^5 \cdot B^3)$ . To prove the equivalence between the constructed instance  $(T_1, T_2, d)$  and  $(\mathcal{S}, T)$ , observe that the big stars in  $T_1$  can only be mapped one-to-one to the big stars in  $T_2$ . The reason for this is that the size of these stars is much greater than the allowed number  $d$  of editions. Then, we have to “separate” the stars in  $\mathcal{S}$  from the big stars in  $T_1$ , and map them into the  $m$  subtrees of  $T_2$ , which correspond to the  $m$  partitions of the 3-PARTITION-instance.  $\square$

### 3 CTS with Bounded Distinct Stars

The NP-hardness of CTS motivates the study of the special case, when the number of distinct stars in  $\mathcal{S}$  is bounded by a constant  $k$ . Let  $S_1, \dots, S_k$  be the distinct stars in  $\mathcal{S}$ . Then,  $\mathcal{S}$  can be denoted by a set of pairs, that is,  $\mathcal{S} = \{(S_1, n_1), \dots, (S_k, n_k)\}$ , where  $n_i$  for  $i = 1, \dots, k$  is the number of copies of  $S_i$  in  $\mathcal{S}$ . Moreover, we use  $|S_i|$  to denote the number of edges in  $S_i$  and assume that  $|S_1| < |S_2| < \dots < |S_k|$ . Clearly,  $|V(T)| = \sum_{i=1}^k (n_i \cdot (|S_i| + 1))$ .

We present in the following a dynamic programming based algorithm solving CTS in polynomial time, if the number of distinct stars is bounded by a constant. The algorithm follows a bottom-up approach to process the vertices in  $T$ . Assume  $T$  is rooted at an arbitrary vertex  $r$ . For each vertex  $v \in V(T)$ ,  $T(v)$  denotes the subtree of  $T$  rooted at  $v$ . During the bottom-up process, we collect some information at every vertex  $v$ . Hereby, we firstly distinguish two cases:  $v$  is “covered” or “free”. We say  $v$  is covered, if  $v$  should be mapped to the center of a star or to a leaf of a star whose center is mapped to a child of  $v$ . A vertex  $v$  is free, if  $v$  should be mapped to a leaf of a star whose center is mapped



**Fig. 2.** The three cases of mapping a node  $v \in V(T)$  to a star  $S_j \in \mathcal{S}$  in CTS with bounded distinct stars

to  $v$ 's parent. Note that a singleton in  $\mathcal{S}$ , that is, an isolated vertex, has only one center but no leaf. Concerning a star  $S$  with only one edge, we say  $v$  is free if the other vertex of  $S$  should be mapped to  $v$ 's parent; otherwise,  $v$  is covered. If  $v$  is covered with  $v$  being mapped to the center of a star  $S$ , then we further distinguish some cases by the size of  $S$ . Hereby, notice that the star  $S$  does not have to be from  $\mathcal{S}$  but has at most  $|S_k|$  leaves. The reason for this is that the parent of  $v$  could be mapped to a leaf of  $S$ . Note that  $S_k$  is the star in  $\mathcal{S}$  with the largest number of leaves.

We use  $C_f$  to denote the case of  $v$  being free and  $C_l$  to denote the case that  $v$  is mapped to a leaf of a star whose center is mapped to a child of  $v$ . Moreover,  $C_c^i$  with  $i = 0, \dots, |S_k|$  denotes the case that  $v$  is mapped to the center of a star with  $i$  leaves, which are mapped to the children of  $v$ . The three cases of mapping vertex  $v$  and its children  $u_1, \dots, u_i$  to a star  $S_j$  are shown on Fig. 2. Altogether, there are  $|S_k| + 3$  many cases to consider.

For each of these cases, we store all possible “realizable configurations” for  $v$ . A *configuration* is defined as a vector  $K = (c_1, \dots, c_k)$  with  $c_i \leq n_i$  for all  $1 \leq i \leq k$ . Each vector  $K$  uniquely represents a subcollection  $\mathcal{S}'$  of  $\mathcal{S}$ , that is,  $\mathcal{S}' = \{(S_1, c_1), \dots, (S_k, c_k)\}$ . The number of all possible configurations is clearly bounded by  $O(|\mathcal{S}|^k)$ . We say a configuration  $K$  is “realizable” at vertex  $v$  with case  $C_f$  (or  $C_l$ ), if there exists a one-to-one mapping  $f$  from  $V(\mathcal{S}')$  to  $V(T(v)) \setminus \{v\}$  (or  $V(T(v))$ ) such that, if  $u, v \in V(\mathcal{S}')$  are adjacent, then  $f(u)$  and  $f(v)$  are adjacent. In the case  $C_c^i$ , configuration  $K$  is realizable, if there exist a size- $(i+1)$  set  $V'$  consisting of  $v$  and  $i$  children of  $v$  and a mapping  $f$  from  $V(\mathcal{S}')$  to  $V(T(v)) \setminus V'$  with the edges in  $\mathcal{S}'$  being preserved. Thus, the given instance is a yes-instance, if and only if at the root  $r$ , the configuration  $(n_1, \dots, n_k)$  is realizable with state  $C_l$  or  $(n_1, \dots, n_i - 1, \dots, n_k)$  is realizable with state  $C_c^{|S_i|}$  for some  $i = 1, \dots, k$ .

**The Algorithm.** In order to compute the realizable configurations for every vertex  $v$  and every case  $\alpha$ , we define the following configuration sets. The set  $\mathcal{K}(v, \alpha)$  should contain all realizable configurations at vertex  $v$  if the case  $\alpha$  applies to  $v$ . Moreover, we define  $\mathcal{K}(v)$  to be the set of the following configurations:

- all configurations in  $\mathcal{K}(v, C_l)$ ,
- for each configuration  $K = (c_1, \dots, c_k)$  in  $\mathcal{K}(v, C_c^i)$  with  $i = |S_\alpha|$  for a star  $S_\alpha \in \mathcal{S}$ , the configuration  $(c_1, \dots, c_\alpha + 1, \dots, c_k)$ .

In other words,  $\mathcal{K}(v)$  contains all configurations, each of which corresponds to a subcollection  $\mathcal{S}'$  such that there exists a mapping  $f$  from  $V(\mathcal{S}')$  to  $V(T_v)$  such that the edges in the stars in  $\mathcal{S}'$  are preserved. We define an addition operation on two configurations  $K^1 = (c_1^1, \dots, c_k^1)$  and  $K^2 = (c_1^2, \dots, c_k^2)$ :  $K_1 + K_2 = (c_1^1 + c_1^2, \dots, c_k^1 + c_k^2)$ , if  $c_i^1 + c_i^2 \leq n_i$  for all  $i = 1, \dots, k$ ; otherwise,  $K_1 + K_2$  is set to an all-0 vector. At the begin,  $\mathcal{K}(v, \alpha)$  and  $\mathcal{K}(v)$  for all vertices  $v$  and all cases  $\alpha$  are set to empty.

At a leaf vertex  $v$ , the cases  $C_l$  and  $C_c^i$  with  $i > 0$  cannot apply and the corresponding configuration sets remain empty. The case  $C_c^0$  can apply, only if  $\mathcal{S}$  contains the singleton. If so, then  $S_1$  is the singleton and both  $\mathcal{K}(v, C_c^0)$  and  $\mathcal{K}(v)$  contain only one configuration  $(c_1, \dots, c_k)$  with  $c_1 = 1$  and  $c_i = 0$  for  $i > 1$ ; otherwise, the two sets are empty. According to the definition of realizable configurations,  $\mathcal{K}(v, C_f)$  contains only the all-0 vector.

Suppose we arrive at an internal vertex  $v$  with  $j$  children  $u_1, u_2, \dots, u_j$ . We distinguish the cases  $C_f$ ,  $C_l$ , and  $C_c^i$ .

**The Free Case  $C_f$ .** By the definition of  $C_f$ , each realizable configuration  $K$  at  $v$  has to correspond to a subcollection  $\mathcal{S}'$  such that each star in  $\mathcal{S}'$  has to be completely mapped to a subtree rooted at one of  $v$ 's children. Moreover, every  $v$ 's child  $u_i$  has to be mapped to either the center or a leaf of a star in  $\mathcal{S}'$  and all other vertices of this star have to be mapped to the vertices in  $T(u_i)$ . Thus, every realizable configuration  $K$  can be “partitioned” into  $j$  configurations  $K^1, \dots, K^j$  such that  $K^i \in \mathcal{K}(u_i)$  for every  $i = 1, \dots, j$  and  $K = K^1 + \dots + K^j$ .

Based on this observation, we compute  $\mathcal{K}(v, C_f)$  as follows: First, for each  $i = 1, \dots, j$ , we first construct  $\mathcal{K}(u_i)$ . To this end, we consider  $\mathcal{K}(u_i, C_c^{|S_\alpha|})$  for each  $\alpha = 1, \dots, k$ ; for each configuration  $K = (c_1, \dots, c_k)$  of this set, we add a configuration  $(c_1, \dots, c_\alpha + 1, \dots, c_k)$  to  $\mathcal{K}(u_i)$ . Finally, all configurations in  $\mathcal{K}(u_i, C_l)$  are added to  $\mathcal{K}(u_i)$ . Then, we initiate  $\mathcal{K}(v, C_f)$  as a set containing only the all-0 vector and iterate from  $i = 1$  to  $i = j$ . For each  $i$ , we perform  $K + K'$  for every pair of  $K$  and  $K'$  with  $K$  being from the old  $\mathcal{K}(v, C_f)$  and  $K' \in \mathcal{K}(u_i)$  and add the result to the new  $\mathcal{K}(v, C_f)$ .

Since the number of configurations is bounded by  $O(|\mathcal{S}|^k)$ , the computation of  $\mathcal{K}(v, C_f)$  is doable in  $O(j \cdot |\mathcal{S}|^{2k})$  time.

**The Covered by Leaf Case  $C_l$ .** In this case, every configuration in  $\mathcal{K}(v, C_l)$  has to correspond to a subcollection  $\mathcal{S}'$ , which contains one star  $S$  with one of  $S$ 's leaves mapping to  $v$  and  $S$ 's center mapping to a child of  $v$ . Note that  $S$  has at least two leaves. The remaining stars of  $\mathcal{S}'$  have to be completely mapped to the subtrees rooted at the children of  $v$ . Therefore, one of  $v$ 's children has to be

of case  $C_c^{|S|-1}$  and others have to be of cases  $C_l$  and  $C_c^i$ . Then,  $\mathcal{K}(v, C_l)$  can be computed as follows:

$$\mathcal{K}(v, C_l) := \bigcup_{S \in \mathcal{S}, |S| > 1} \left( \bigcup_{i=1, \dots, j} \mathcal{K}^{S,i}(v, C_l) \right) ,$$

where  $\mathcal{K}^{S,i}(v, C_l)$  contains all realizable configurations with  $v$  being mapped to a star  $S$  whose central is mapped to  $u_i$ . The set  $\mathcal{K}^{S,i}(v, C_l)$  can be computed in a similar way as  $\mathcal{K}(v, C_f)$ : First, we compute  $\mathcal{K}(u_{i'})$  for all  $i' \neq i$ , as in the  $C_f$ -case. We initialize  $\mathcal{K}^{S,i}(v, C_l)$  as a set containing only the all-0 vector and iterate over all  $i = 1, \dots, j$  as in the  $C_f$ -case. The only exception is that the set  $\mathcal{K}(u_i)$  is replaced by  $\mathcal{K}(u_i, C_c^{|S|-1})$ . Finally, we increase the entry of  $\mathcal{K}^{S,i}(v, C_l)$  corresponding to  $S$  by one, since one copy of  $S$  is now completely mapped in  $T(v)$ .

Note that we need only to compute  $\mathcal{K}^{S,i}(v, C_l)$  for distinct stars  $S \in \mathcal{S}$ . Since the computation of  $\mathcal{K}^{S,i}(v, C_l)$  is basically the same as the one of  $\mathcal{K}(v, C_f)$ , the overall time for computing  $\mathcal{K}(v, C_l)$  is bounded by  $O(k \cdot j^2 \cdot |\mathcal{S}|^{2k})$ .

**The Covered by Center Case  $C_c^i$ .** Now,  $v$  has to be mapped to the center of a star  $S$  with  $0 \leq i = |S| \leq |S_k|$  and  $S$ 's leaves have to be mapped to some of  $u_1, \dots, u_j$ . Note that  $S$  does not necessarily appear in  $\mathcal{S}$  and  $|S| \leq j$ . Similarly to the  $C_f$ - and  $C_l$ -cases, all realizable configurations in  $\mathcal{K}(v, C_c^{|S|})$  correspond to subcollections  $\mathcal{S}'$  whose stars can be partitioned to  $j$  subsets, each being completely mapped to a subtree rooted at  $v$ 's children. The only difference here is that, in order to map  $v$  to the center of  $S$ , there must be  $|S|$  children of  $v$  which are “free”, that is, of case  $C_f$ . Thus, it remains to compute all possible realizable configurations for every distinct star  $S \in \mathcal{S}$  with  $|S| \leq j$ . We apply here again a dynamic programming approach. Assume that  $\mathcal{K}(u_i)$  for all  $i = 1, \dots, j$  have already been computed, which can be done as described in the  $C_f$ -case.

We define a table  $\mathcal{T}$  of size at most  $j(j+1)/2$ . The entry  $\mathcal{T}[\alpha, \beta]$  with  $1 \leq \alpha \leq j$  and  $0 \leq \beta \leq |S| \leq j$  stores all possible realizable configurations, that we can have in the subtrees  $T(u_1), \dots, T(u_\alpha)$ , if exactly  $\beta$  many roots of these trees are set to the free case. Clearly, we consider only  $\mathcal{T}[\alpha, \beta]$  with  $\beta \leq \alpha$ .

The computation of the first row of  $\mathcal{T}$  is trivial. By definition,  $\mathcal{T}[1, 0]$  is clearly set to  $\mathcal{K}(u_1)$ , while  $\mathcal{T}[1, 1]$  means the same as  $\mathcal{K}(u_1, C_f)$ . The entries  $\mathcal{T}[\alpha, \beta]$  with  $\alpha > 1$  can be computed as follows:

$$\mathcal{T}[\alpha, 0] := \mathcal{T}[\alpha - 1, 0] + \mathcal{K}(u_\alpha) ,$$

$$\mathcal{T}[\alpha, \beta] := (\mathcal{T}[\alpha - 1, \beta] + \mathcal{K}(u_\alpha)) \cup (\mathcal{T}[\alpha - 1, \beta - 1] + \mathcal{K}(u_\alpha, C_f)) .$$

By the definition, the entry  $\mathcal{T}[j, |S|]$  then contains all possible realizable configurations under the condition that  $v$  and  $|S|$  many its children are mapped to  $S$ . Thus, we set  $\mathcal{K}(v, C_c^{|S|}) := \mathcal{T}[j, |S|]$ .

With the same argument as in the  $C_f$ -case, the operations between configuration sets can be done in  $O(|\mathcal{S}|^{2k})$  time. Thus, the time for computing  $\mathcal{T}$  is bounded by  $O(j^2 \cdot |\mathcal{S}|^{2k})$ .

**Theorem 3.** *If the number of distinct stars in  $\mathcal{S}$  is bounded by a constant, then CTS can be solved in polynomial time.*

*Proof.* The correctness of the dynamic programming algorithm follows from the correctness of the computation for leaves and the three cases of internal vertices. From the time analysis of the cases, we can easily derive an overall  $O(|V(T)|^2 \cdot k \cdot |V(\mathcal{S})|^{2k})$ -time bound for the dynamic programming algorithm, which is polynomial in  $|V(T)|$ , if  $k$  is bounded by a constant.  $\square$

## 4 Conclusion

We studied the tree edit distance problem with edge deletions and edge insertions as edit operations and proved its NP-hardness even for trees with bounded diameters. To this end, we reformulated a restrictive case of the problem as COVERING TREE WITH STARS (CTS) and showed that it is NP-hard. Moreover, we show that CTS can be solved in polynomial time when the number of distinct stars is bounded by  $k$ . To this end, we presented a dynamic programming algorithms running in  $O(|V(T)|^2 \cdot k \cdot |V(\mathcal{S})|^{2k})$  time. It is still open whether CTS is fixed-parameter tractable with  $k$  as parameter, that is, whether there is an algorithm for CTS with time  $f(k) \cdot |V(T)|^{O(1)}$  for a function  $f$ . Moreover, it is also interesting to find out other constraints, which can lead to polynomial-time solvability of TED-ID.

## References

1. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty Years of Graph Matching in Pattern Recognition. International Journal of Pattern Recognition and Artificial Intelligence (2004)
2. Gao, X., Xiao, B., Tao, D., Li, X.: A survey of graph edit distance. Pattern Analysis & Applications 13(1), 113–129 (2010)
3. Bunke, H., Riesen, K.: Graph Edit Distance – Optimal and Suboptimal Algorithms with Applications, pp. 113–143. Wiley-VCH Verlag GmbH & Co. KGaA (2009)
4. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman, San Francisco (1979)
5. Akutsu, T., Fukagawa, D., Takasu, A., Tamura, T.: Exact algorithms for computing the tree edit distance between unordered trees. Theor. Comput. Sci. 412, 352–364 (2011)
6. Bille, P.: A survey on tree edit distance and related problems. Theor. Comput. Sci. 337(1-3), 217–239 (2005)
7. Akutsu, T.: Tree Edit Distance Problems: Algorithms and Applications to Bioinformatics. IEICE Transactions on Information and Systems 93, 208–218 (2010)
8. Demaine, E.D., Mozes, S., Rossman, B., Weimann, O.: An optimal decomposition algorithm for tree edit distance. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 146–157. Springer, Heidelberg (2007)
9. Zhang, K., Statman, R., Shasha, D.: On the editing distance between unordered labeled trees. Information Processing Letters 42(3), 133–139 (1992)

10. Zhang, K., Jiang, T.: Some MAX SNP-hard results concerning unordered labeled trees. *Information Processing Letters* 49(5), 249–254 (1994)
11. Akutsu, T., Fukagawa, D., Halldórsson, M.M., Takasu, A., Tanaka, K.: Approximation and parameterized algorithms for common subtrees and edit distance between unordered trees. *Theoretical Computer Science* 470, 10–22 (2013)
12. Blin, G., Sikora, F., Vialette, S.: Querying graphs in protein-protein interactions networks using feedback vertex set. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 7(4), 628–635 (2010)
13. Natanzon, A., Shamir, R., Sharan, R.: Complexity classification of some edge modification problems. *Discrete Applied Mathematics* 113(1), 109–128 (2001)
14. Sharan, R.: Graph modification problems and their applications to genomic research. PhD thesis, School of Computer Science (2002)
15. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: *The Design and Analysis of Computer Algorithms*, 1st edn. Addison-Wesley Longman Publishing Co., Inc., Boston (1974)

# A Polynomial Time Approximation Scheme for the Closest Shared Center Problem

Weidong Li<sup>1</sup>, Lusheng Wang<sup>2</sup>, and Wenjuan Cui<sup>2</sup>

<sup>1</sup> Department of Atmospheric Science,  
Yunnan University, Kunming, P.R. China

<sup>2</sup> Department of Computer Science,  
City University of Hong Kong, Hong Kong

weidong@ynu.edu.cn, cswangl@cityu.edu.hk, wenjuacui2@student.cityu.edu.hk

**Abstract.** Mutation region detection is the first step of searching for a disease gene and has facilitated the identification of several hundred human genes that can harbor mutations leading to a disease phenotype. Recently, the *closest shared center* problem (CSC) was proposed as a core to solve the mutation region detection problem when the pedigree is not given [9]. A ratio-2 approximation algorithm was proposed for the closest shared center problem. In this paper, we will design a polynomial time approximation scheme for this problem.

## 1 Introduction

Mutation region detection is the first step of searching for a disease gene. Linkage analysis has facilitated the identification of several hundred human genes that can harbor mutations leading to a disease phenotype. The fundamental problem in linkage analysis is to identify regions whose allele is shared by all affected members but by none unaffected family members.

There are mainly two approaches for linkage analysis, probabilistic approach and deterministic approach. Most methods for linkage analysis are designed for families with clearly given pedigrees. In probabilistic approaches, recombinant rates are estimated in a way to maximize the likelihood of the observed data [1,4,6]. In deterministic approaches, the aim is to minimize the total number of recombinants to infer the input genotype data so that all diseased individuals share a segment that is shared by none of the normal individuals [2,8].

Recently, there are some work on the important case, where the sampled individuals are closely related, but the pedigree is not given [9,10,11]. This situation happens very often when the individuals share a common ancestor 6 or more generations ago.

Here we assume that for each input individual we know the genotype data  $g$ , over a chromosome. A genotype segment is a string on alphabet  $\{0, 1, 2\}$ . A haplotype segment is a binary string on alphabet  $\{0, 1\}$ . A haplotype pair for a genotype segment  $g$  is a pair  $(h, h')$  of haplotype segments of the same length as  $g$  such that the following conditions hold for every position  $q$ :

- If  $g$  has a 0 or 1 at position  $q$ , then both  $h$  and  $h'$  have the same letter as  $g$  does at position  $q$ .
- If  $g$  has a 2 at position  $q$ , then one of  $h$  and  $h'$  has a 0 at position  $q$  while the other has a 1 at position  $q$ .

The general problem is as follows: We are given two sets of genotypes on the whole chromosome  $D = \{G_1, G_2, \dots, G_k\}$  and  $N = \{G_{k+1}, G_{k+2}, \dots, G_n\}$ , where the  $k$  genotypes in  $D$  are from diseased individuals and the  $n - k$  genotypes in  $N$  are from normal individuals. The  $n$  individuals in  $D$  and  $N$  are relatives (in the same hidden pedigree). The objective is to detect the *mutation regions* on the chromosome such that all the diseased individuals share a common haplotype segment on the region and none of the normal individuals has such a common haplotype segment on the region. Here, we study monogenic dominant diseases, which is caused by one single gene. The diseases can happen once the faulty gene appears on a single haplotype. Therefore, the mutation region is a region on a single haplotype containing the faulty gene. On each region, each individual has two haplotype segments. As indicated in [10], if we know the haplotype of each input individual over the chromosome, the *true mutation regions* can be computed by finding the haplotype segments which are shared by all the diseased individuals but by none of the normal ones. Therefore, to solve the problem, the key issue is to infer the haplotype segments from the genotype data.

The task of inferring the haplotype of each individual over the whole chromosome is difficult. To solve the general problem, Ma et al. [9] propose a method that makes use of the known haplotypes of similar population as reference to infer haplotypes. The main problem they considered is the mutation region detection problem, where a region on a chromosome, denoted by  $[a, b]$ , is a set of consecutive SNP sites (positions) starting at position  $a$  and ending at position  $b$ . The mutation region detection problem is as follows: We are given three sets  $D = \{g_1, g_2, \dots, g_k\}$ ,  $N = \{g_{k+1}, g_{k+2}, \dots, g_n\}$ , and  $H = \{h_1, h_2, \dots, h_m\}$ , where  $D$  consists of diseased individuals represented by their genotype data on a whole chromosome  $C$ ,  $N$  consists of normal individuals represented by their genotype data on  $C$ , and  $H$  consists of confirmed haplotype data on  $C$  of some individuals in the same (or similar) population. The objective is to find the true mutation regions of  $C$ . Here, a true mutation region of  $C$  means a consecutive portion of  $C$  where all the diseased individuals share a common haplotype segment that is shared by none of the normal individuals. The true mutation regions defined here are based on the haplotype segments of all individuals. If we know the haplotype segments of all the individuals, the true mutation regions can be easily computed.

To solve the mutation region detection problem, Ma et al. [9] proposed to decompose the whole chromosome into a set of disjoint regions and test if a region belongs to the true mutation region. To test if a region belongs to the true mutation region, Ma et al. [9] proposed a mathematical model, the *shared center problem*. They showed that the shared center problem was  $NP$ -complete and then presented a 2-approximation algorithm for the closest shared center

(CSC, for short) problem which is the minimization version of the shared center problem.

The formal definition of CSC is presented as follows. We are given three sets  $D = \{g_1, g_2, \dots, g_k\}$ ,  $N = \{g_{k+1}, g_{k+2}, \dots, g_n\}$ , and  $H = \{h_1, h_2, \dots, h_m\}$ , where  $g_i$  is a genotype of length  $L$  for  $i = 1, 2, \dots, n$ , and  $h_j$  is a haplotype of length  $L$  for  $j = 1, 2, \dots, m$ . For a genotype  $g$  in  $D \cup N$ , the letter  $g[i]$  of  $g$  at position  $i$  can be 0, 1, or 2. Thus,  $g = g[1]g[2]\dots g[L]$ . For a haplotype  $h$  in  $H$ , the letter  $h[i]$  of  $h$  at position  $i$  can be 0 or 1. Thus,  $h = h[1]h[2]\dots h[L]$ .

For convenience, we say that a position of  $g \in D \cup N$  is *decided* if the letter of  $g$  at the position is 0 or 1, and is *undecided* otherwise. A *haplotype pair* for  $g_i \in D \cup N$  is a pair  $(h_{i,1}, h_{i,2})$  of haplotypes satisfying the following conditions:

- 1) The letter of  $g_i$  at each decided position  $q$  is the same as the letters of both  $h_{i,1}$  and  $h_{i,2}$  at the same position, i.e.,  $g_i[q] = a \in \{0, 1\}$  indicates that the inferred haplotypes  $h_{i,1}$  and  $h_{i,2}$  of  $g_i$  both satisfy  $h_{i,1}[q] = h_{i,2}[q] = a$ .
- 2) For each undecided position of  $g_i$ , one of  $h_{i,1}$  and  $h_{i,2}$  has a 0 at the position while the other has a 1 at the position.

For the genotype set  $D$ , we define three sets of positions as follows:

- The set of conflicting positions associated with  $D$  consists of all positions  $q$  such that  $q$  is a decided position of two distinct  $g_i \in D$  and  $g_j \in D$  but the letters of  $g_i$  and  $g_j$  at position  $q$  differ. To assure the existence of a feasible solution, we assume that this set is empty.
- The set  $\bar{U}$  of decided positions associated with  $D$  consists of all positions  $q$  such that  $q$  is a decided position of *at least* one genotype segment in  $D$ .
- The set  $U$  of undecided positions associated with  $D$  consists of all positions  $q$  such that  $q$  is an undecided position for *all* genotype segment in  $D$ .

For two haplotypes  $h$  and  $h'$ , let  $d(h, h')$  denote the *Hamming distance* between  $h$  and  $h'$ . Let  $P = \{j_1, j_2, \dots, j_l\}$  satisfying  $1 \leq j_1 < j_2 < \dots < j_l \leq L$  be a set of positions. We write  $h|_P$  to denote the haplotype segment  $h[j_1]h[j_2]\dots h[j_l]$ . We also use  $d^P(h, h')$  to denote  $d(h|_P, h'|_P)$ . For two positive integers  $a, b$  with  $a < b$ ,  $[a..b]$  denotes the integer set  $\{a, a + 1, \dots, b - 1, b\}$ .

Let  $\mathcal{I} = (D, N, H)$  be the input instance. Given an integer  $d \in [0..L]$  (referred to as the *radius*), a feasible solution for  $\mathcal{I}$  consists of a haplotype  $s$  of length  $L$  (referred to as the *center haplotype*), an index  $p \in [1..m]$  (referred to as the *center index*), and a haplotype pair  $(h_{i,1}, h_{i,2})$  for each genotype  $g_i \in D \cup N$  such that the following conditions hold:

- C1  $d(s, h_p) \leq d$ .
- C2 for each  $i \in [1..n]$ ,  $h_{i,1} = s$  and there is an integer  $l_i \in [1..m]$  such that  $d(h_{i,2}, h_{l_i}) \leq d$ .
- C3 for each  $i \in [k+1..n]$  and for each  $j \in \{1, 2\}$ , the following hold:  
 C3a. There is an integer  $l_{i,j} \in [1..m] \setminus \{p\}$  with  $d(h_{i,j}, h_{l_{i,j}}) \leq d$ .  
 C3b.  $h_{i,j}|_{\bar{U}} \neq s|_{\bar{U}}$ , i.e., there is at least one position  $q$  in  $\bar{U}$  at which the letters of  $h_{i,j}$  and  $s$  differ.

For a given integer  $d$ , if there is a feasible solution for  $\mathcal{I}$ ,  $d$  is called *valid radius*. The CSC problem is to find the minimum valid radius for instance  $\mathcal{I}$ . In this

paper, we design a polynomial time approximation scheme for CSC, i.e., for any fixed number  $\epsilon \in (0, 1)$ , our algorithm can find a feasible solution with objective value no more than  $(1 + \epsilon)d^*$ , where  $d^*$  is the optimal objective value of the input instance for the CSC problem.

## 2 Preliminaries

As in [9], we assume that  $L$  is a valid radius for instance  $\mathcal{I}$ , otherwise, there is no feasible solution. Let  $d^*$  be the optimal radius for  $\mathcal{I}$ . Given any fixed number  $\epsilon \in (0, 1)$ , for each possible center index  $p \in [1..m]$ , our method is to find a feasible solution with objective value no more than  $(1 + \epsilon)d_p$  and output the solution with minimum radius, where  $d_p$  is the optimal value for the restricted instance satisfying that  $p$  is the center index. As  $d^* = \min\{d_p | p \in [1..m]\}$ , the objective value of the outputted solution is no more than  $(1 + \epsilon)d^*$ . Hence, without loss of generality, we assume that the center index  $p$  is known.

Here, we first focus on designing an approximation scheme to find a haplotype pair  $(h_{i,1}, h_{i,2})$  for each genotype  $g_i \in D$  satisfying Conditions C1 and Conditions C2. If there is a haplotype  $s$  of length  $L$  shared by the genotypes in  $D$ , then the letter of  $s$  at each position  $q \in \bar{U}$  can be uniquely fixed according to the following rules:

Rule 1. If some genotype in  $D$  is 0 at position  $q$  and each of the other segments in  $D$  is 0 or 2 at position  $q$ , then the letter of  $s$  at position  $q$  is 0.

Rule 2. If some genotype in  $D$  is 1 at position  $q$  and each of the other segments in  $D$  is 1 or 2 at position  $q$ , then the letter of  $s$  at position  $q$  is 1.

Thus, the letter of  $s$  at each decided position  $q \in \bar{U}$  is known. For each  $i \in [1..k]$  and  $q \in \bar{U}$ , if  $g_i[q] = 2$ , set  $h_{i,2}[q] = 1 - s[q]$ , otherwise set  $h_{i,2}[q] = s[q]$ . Thus, we obtain the letters of  $h_{i,2}|_{\bar{U}}$ . The remaining task is to find the letters of  $s$  at each position  $q \in U$ . Without loss of generality, we assume that  $U = \{1, 2, \dots, u\}$ , where  $u = |U|$ .

For a haplotype  $h$  of length  $|h|$ ,  $\bar{h}$  denotes the complement haplotype of  $h$ , where  $\bar{h}[i] \neq h[i]$  for every  $i \in [1..|h|]$ . Since  $s[q] = h_{i,1}[q] \neq h_{i,2}[q]$  for each  $q \in U$ , we have  $\bar{s}|_U = h_{i,2}|_U$ . Thus,  $d(h_{i,2}, h_{l_i}) = d^U(h_{i,2}, h_{l_i}) + d^{\bar{U}}(h_{i,2}, h_{l_i}) = d^U(\bar{s}, h_{l_i}) + d^{\bar{U}}(h_{i,2}, h_{l_i})$ , for each  $i \in [1..k]$ . Also, we have  $d(s, h_p) = d^U(s, h_p) + d^{\bar{U}}(s, h_p) = d^U(\bar{s}, h_p) + d^{\bar{U}}(s, h_p)$ . Therefore, the CSC problem can be formulated as follows.

$$\begin{cases} \min & d \\ d^U(\bar{s}, \bar{h}_p) \leq d - d^{\bar{U}}(s, h_p); & \text{Condition C1} \\ d^U(\bar{s}, h_{l_i}) \leq d - d^{\bar{U}}(h_{i,2}, h_{l_i}), & \forall i \in [1..k]. \quad \text{Condition C2} \end{cases} \quad (1)$$

In the next section, we will design a polynomial time approximation scheme (PTAS) that can find a feasible solution for (1) with objective value no more than  $(1 + \epsilon)d_p$  for any constant  $\epsilon \in (0, 1)$ , where  $d_p$  is the optimal objective value for (1).

### 3 Approximation Scheme for $D$

For convenience, let  $s$  and  $h_{l_i}$  ( $i \in [1..k]$ ) be the optimal solution for (1) from now on. The main difficulty for solving (1) is that the indices  $p$  and  $l_i$  for each  $i \in [1..k]$  are not known. For any positive integer  $r \geq 2$ , consider  $r$  haplotypes  $h_{i_1}, h_{i_2}, \dots, h_{i_r} \in \{\bar{h}_p\} \cup H$ . Let  $Q_{i_1, i_2, \dots, i_r} \subseteq U$  be the set of positions in  $U$  where  $h_{i_1}, h_{i_2}, \dots, h_{i_r}$  agree, and  $P_{i_1, i_2, \dots, i_r} = U \setminus Q_{i_1, i_2, \dots, i_r}$ . Firstly, we use  $h_{i_1}|_{Q_{i_1, i_2, \dots, i_r}}$  to approximate  $\bar{s}|_{Q_{i_1, i_2, \dots, i_r}}$ . Then, use a random sampling technique to find a  $h'_{l_i} \in H$  which is close to  $h_{l_i}$ . After replacing  $h_{l_i}$  by  $h'_{l_i}$ , we reformulate (1) as a linear programming and use the randomized rounding approach to find a string of length  $|P_{i_1, i_2, \dots, i_r}|$  to approximate  $\bar{s}|_{P_{i_1, i_2, \dots, i_r}}$ .

#### 3.1 Approximate $\bar{s}|_{Q_{i_1, i_2, \dots, i_r}}$

We will prove that there exist  $r$  haplotypes  $h_{i_1}, h_{i_2}, \dots, h_{i_r}$  such that  $h_{i_1}$  is close to  $\bar{s}$  at the positions in  $Q_{i_1, i_2, \dots, i_r}$ .

**Lemma 1.** Let  $h_{l_0} = \bar{h}_p$ . For any constant  $r \in [2..k]$ , there exist  $r$  haplotypes  $h_{i_1}, h_{i_2}, \dots, h_{i_r}$  in  $\{h_{l_0}, h_{l_1}, \dots, h_{l_k}\}$  such that, for each  $i \in [0..k]$ ,

$$d^Q(h_{i_1}, h_{l_i}) - d^Q(\bar{s}, h_{l_i}) \leq \frac{1}{2^{r-1}}d_p, \quad \text{where } Q = Q_{i_1, i_2, \dots, i_r}.$$

**Proof.** Choose  $h_{i_1} = h_{l_0}$ . Let  $T = \{q \in U | h_{i_1}[q] \neq \bar{s}[q]\}$  be the set of positions where  $h_{i_1}$  and  $\bar{s}$  are different. Following Condition C1 in (1), we have  $|T| \leq d_p$ . We use the following method to select the remaining (at most)  $r - 1$  indices.

Step 1. For the remaining haplotypes in  $\{h_{l_1}, \dots, h_{l_k}\}$  do

Step 2. Select the haplotype  $h_{l_i}$  satisfying  $d^T(h_{i_1}, h_{l_i}) \geq \frac{|T|}{2}$  and then set  $T = T \setminus \{q \in T | h_{i_1}[q] \neq h_{l_i}[q]\}$ . If  $T = \emptyset$ , stop, otherwise, goto Step 3.

Step 3. If we have selected  $r$  haplotypes including  $h_{i_1}$ , stop; else, goto Step 1.

Clearly, the above procedure will stop after at most  $r - 1$  iterations. Since each time the size of  $T$  is reduced by at least half, the size of  $T$  after  $r - 1$  iterations is at most  $\frac{1}{2^{r-1}}d_p$ . Note that the selected haplotypes agree at each position  $q \in T$  and  $h_{i_1}[q] = \bar{s}[q]$  for each  $q \in Q \setminus T$  by the definition of  $T$ . For each  $i \in [0..k]$ , we have  $d^Q(h_{i_1}, h_{l_i}) - d^Q(\bar{s}, h_{l_i}) = d^T(h_{i_1}, h_{l_i}) - d^T(\bar{s}, h_{l_i}) \leq d^T(h_{i_1}, h_{l_i}) \leq |T| \leq \frac{1}{2^{r-1}}d_p$ .

Next, we will prove that there exists a haplotype satisfying the condition at Step 2. If not, construct a new haplotype  $s^*$  by modifying  $\bar{s}$ : for each  $q \in T$ , set  $s^*[q] = h_{i_1}[q]$ . By assumption, for each unselected haplotype  $h_{l_i}$  at Step 2, we have  $d^T(s^*, h_{l_i}) = d^T(h_{i_1}, h_{l_i}) < \frac{|T|}{2}$ . For each  $q \in T$ , if  $s^*[q] = h_{l_i}[q]$ , then  $\bar{s}[q] \neq h_{l_i}[q]$ . Thus, we have  $d^T(\bar{s}, h_{l_i}) > \frac{|T|}{2}$ . Hence,  $d^T(s^*, h_{l_i}) < d^T(\bar{s}, h_{l_i})$ , which implies that  $d^U(s^*, h_{l_i}) < d^U(\bar{s}, h_{l_i})$ . By the definition of  $T$ ,  $d^T(s^*, h_{l_i}) = 0$  holds for each selected haplotype  $h_{l_i}$ . Therefore, for each haplotype  $h_{l_i}$  (selected or unselected),  $d^U(s^*, h_{l_i}) < d^U(\bar{s}, h_{l_i})$ . This contradicts the optimality of  $\bar{s}$ .

Although we do not know  $h_{l_i}$ , we can try all possible size  $r$  haplotypes (admits repetition) in  $\{\bar{h}_p\} \cup H$  within  $O((m + 1)^r)$  time. Thus, we assume that the  $r$  haplotypes  $h_{i_1}, h_{i_2}, \dots, h_{i_r}$  satisfying Lemma 1 are known. For convenience, let  $Q = Q_{i_1, i_2, \dots, i_r}$  and  $P = U \setminus Q$ .

**Lemma 2.**  $|P| \leq rd_p$ .

**Proof.** For each  $q \in P$ , by the definition of  $P$ , there exists some  $h_{i_j}$  ( $j \in [1..r]$ ) such that  $h_{i_j}[q] \neq \bar{s}[q]$ . Since  $d^U(\bar{s}, h_{i_j}) \leq d_p$ , each  $h_{i_j}$  contributes at most  $d_p$  positions in  $P$ . Thus,  $|P| \leq rd_p$ .

### 3.2 Approximate $h_{l_i}$ by Using the Random Sampling Approach

The following lemma is very useful throughout this section.

**Lemma 3.** [7] Let  $X_1, X_2, \dots, X_n$  be  $n$  independent random 0-1 variables, where  $X_i$  takes 1 with probability  $p_i$ ,  $0 < p_i < 1$ . Let  $X = \sum_{i=1}^n E[X_i]$ . Then for any  $0 < \delta \leq 1$ ,

$$\begin{aligned}\Pr(X > \mu + \delta n) &\leq \exp\left(-\frac{1}{3}n\delta^2\right), \\ \Pr(X < \mu - \delta n) &\leq \exp\left(-\frac{1}{2}n\delta^2\right).\end{aligned}$$

Although we have found a string  $h_{i_1}|_Q$  which is close to  $\bar{s}|_Q$ , we know nothing about  $\bar{s}|_P$ . Also, we do not know  $h_{l_i}$  for each  $i \in [1..k]$ . We will use the random sampling strategy to find a  $h_{l'_i}$  which is near to  $h_{l_i}$ . We randomly pick  $\lceil 4/\delta^2 \ln(mk) \rceil$  positions from  $P$ . Suppose the multiset of these random positions is  $R$ . By trying  $2^{|R|} = O((mk)^{\frac{4}{\delta^2}})$  possible strings, we can assume that we know  $\bar{s}|_R$ . For each  $i \in [1..k]$ , we find the haplotype  $h_{l'_i} \in H$  such that

$$f(h_{l'_i}) = d^R(\bar{s}, h_{l'_i}) \cdot |P|/|R| + d^Q(h_{i_1}, h_{l'_i}) + d^{\bar{U}}(h_{i,2}, h_{l'_i})$$

is minimized.

To prove that  $h_{l'_i}$  is a good approximation to  $h_{l_i}$ , we introduce a binary string  $s^*$  of length  $u$  satisfying  $s^*|_P = \bar{s}|_P$  and  $s^*|_Q = h_{i_1}|_Q$ .

**Lemma 4.** With probability at most  $(mk)^{-1}$ , there is a haplotype  $h_{l'_i} \in H$  ( $i \in [1..k]$ ) satisfying

$$f(h_{l'_i}) \leq d^U(s^*, h_{l'_i}) + d^{\bar{U}}(h_{i,2}, h_{l'_i}) - \delta|P|. \quad (2)$$

With probability at most  $(mk)^{-\frac{1}{3}}$ , there is a haplotype  $h_{l_i}$  ( $i \in [1..k]$ ) satisfying

$$f(h_{l_i}) \geq d^U(s^*, h_{l_i}) + d^{\bar{U}}(h_{i,2}, h_{l_i}) + \delta|P|. \quad (3)$$

**Proof.** Let  $\rho = |P|/|R|$  and then  $f(h) = \rho \cdot d^R(\bar{s}, h) + d^Q(h_{i_1}, h) + d^{\bar{U}}(h_{i,2}, h)$  for each  $h \in H$ . We will consider  $f(h_{l'_i})$  first. Since  $R$  is a set of randomly independently selected positions,  $d^R(\bar{s}, h_{l'_i})$  is the sum of  $|R|$  independent random 0-1 variables  $\sum_{l=1}^{|R|} X_l$ , where  $X_l = 1$  indicates a mismatch between  $\bar{s}$  and  $h_{l'_i}$  at the  $l$ th position in  $R$ .

Let  $\mu' = E[d^R(\bar{s}, h_{l'_i})]$ . Clearly,  $\mu' = d^P(\bar{s}, h_{l'_i}) \cdot |R|/|P| = d^P(\bar{s}, h_{l'_i})/\rho$ . By Lemma 3, we have

$$\begin{aligned} & \mathbf{Pr}(f(h_{l'_i}) \leq d^U(s^*, h_{l'_i}) + d^{\bar{U}}(h_{i,2}, h_{l'_i}) - \delta|P|) \\ &= \mathbf{Pr}(\rho \cdot d^R(\bar{s}, h_{l'_i}) + d^Q(h_{i,1}, h_{l'_i}) + d^{\bar{U}}(h_{i,2}, h_{l'_i}) \leq d^U(s^*, h_{l'_i}) + d^{\bar{U}}(h_{i,2}, h_{l'_i}) - \delta|P|) \\ &= \mathbf{Pr}(\rho \cdot d^R(\bar{s}, h_{l'_i}) + d^Q(s^*, h_{l'_i}) \leq d^U(s^*, h_{l'_i}) - \delta|P|) \\ &= \mathbf{Pr}(d^R(\bar{s}, h_{l'_i}) \leq d^P(\bar{s}, h_{l'_i})/\rho - \delta|P|/\rho) \\ &= \mathbf{Pr}(d^R(\bar{s}, h_{l'_i}) \leq \mu' - \delta|R|) \\ &\leq \exp(-\frac{1}{2}\delta^2|R|) \\ &\leq (mk)^{-2}, \end{aligned}$$

where the second equality follows the fact  $s^*|_Q = h_{i,1}|_Q$ . Since  $|H| = m$ , there are at most  $m$  possible haplotypes for each  $i \in [1..k]$ . Considering all the  $mk$   $f(h_{l'_i})$ 's, the probability that there is a  $h_{l'_i}$  satisfying (2) is at most  $(mk) \times (mk)^{-2} = (mk)^{-1}$ .

Similarly, we have

$$\mathbf{Pr}(f(h_{l_i}) \geq d^U(s^*, h_{l_i}) + d^{\bar{U}}(h_{i,2}, h_{l_i}) + \delta|P|) \leq (mk)^{-\frac{4}{3}}.$$

Considering all the  $mk$   $h_{l_i}$ 's, the probability that a  $h_{l_i}$  satisfies (3) is at most  $(mk) \times (mk)^{-\frac{4}{3}} = (mk)^{-\frac{1}{3}}$ . Thus, the lemma holds.

**Lemma 5.** With high probability,  $d^U(s^*, h_{l'_i}) + d^{\bar{U}}(h_{i,2}, h_{l'_i}) \leq (1 + \frac{1}{2^{r-1}})d_p + 2\delta|P|$  for each  $i \in [1..k]$ .

**Proof.** For any two strings  $t$  and  $t'$ , we have  $d^P(t, t') + d^Q(t, t') = d^U(t, t')$ . Combining Lemma 1, for any  $i \in [1..k]$ , we have

$$\begin{aligned} d^U(s^*, h_{l_i}) + d^{\bar{U}}(h_{i,2}, h_{l_i}) &= d^P(s^*, h_{l_i}) + d^Q(s^*, h_{l_i}) + d^{\bar{U}}(h_{i,2}, h_{l_i}) \\ &= d^P(\bar{s}, h_{l_i}) + d^Q(h_{i,1}, h_{l_i}) + d^{\bar{U}}(h_{i,2}, h_{l_i}) \\ &\leq d^P(\bar{s}, h_{l_i}) + d^Q(\bar{s}, h_{l_i}) + \frac{1}{2^{r-1}}d_p + d^{\bar{U}}(h_{i,2}, h_{l_i}) \quad (4) \\ &= d^U(\bar{s}, h_{l_i}) + d^{\bar{U}}(h_{i,2}, h_{l_i}) + \frac{1}{2^{r-1}}d_p \\ &\leq (1 + \frac{1}{2^{r-1}})d_p, \end{aligned}$$

where the last inequality follows the fact that  $d^U(\bar{s}, h_{l_i}) + d^{\bar{U}}(h_{i,2}, h_{l_i}) \leq d_p$ . Thus, we only need to prove that with high probability,

$$d^U(s^*, h_{l'_i}) + d^{\bar{U}}(h_{i,2}, h_{l'_i}) \leq d^U(s^*, h_{l_i}) + d^{\bar{U}}(h_{i,2}, h_{l_i}) + 2\delta|P|. \quad (5)$$

By Lemma 4, with probability at least  $1 - (mk)^{-1} - (mk)^{-\frac{1}{3}}$ , we have

$$f(h_{l'_i}) \geq d^U(s^*, h_{l'_i}) + d^{\bar{U}}(h_{i,2}, h_{l'_i}) - \delta|P|,$$

and

$$f(h_{l_i}) \leq d^U(s^*, h_{l_i}) + d^{\bar{U}}(h_{i,2}, h_{l_i}) + \delta|P|.$$

Since we choose  $h_{l'_i}$  such that  $f(h_{l'_i}) \leq f(h_{l_i})$  for each  $i \in [1..k]$ , we obtain

$$d^U(s^*, h_{l'_i}) + d^{\bar{U}}(h_{i,2}, h_{l'_i}) \leq d^U(s^*, h_{l_i}) + d^{\bar{U}}(h_{i,2}, h_{l_i}) + 2\delta|P|.$$

Combining (4), for each  $i \in [1..k]$ , we have

$$d^U(s^*, h_{l'_i}) + d^{\bar{U}}(h_{i,2}, h_{l'_i}) \leq (1 + \frac{1}{2^{r-1}})d_p + 2\delta|P|.$$

This completes the lemma.

**Corollary 6.** In polynomial time, we can find a  $h_{l'_i}$  that satisfies  $d^U(s^*, h_{l'_i}) + d^{\bar{U}}(h_{i,2}, h_{l'_i}) \leq (1 + \frac{1}{2^{r-1}})d_p + 2\delta|P|$  for each  $i \in [0..k]$ .

**Proof.** The algorithm in the proof of Lemma 5 can be derandomized by the method in [7] (or [3] for details).

When we choose large  $r$  and small  $\delta$ , Corollary 6 implies that  $h_{l'_i}$  is a good approximation for  $h_{l_i}$ . From now on, we can assume that  $h_{l'_i}$  satisfying the condition in Corollary 6 is known for each  $i \in [1..k]$ .

### 3.3 Approximate $\bar{s}|_P$ by Using the Randomized Rounding Approach

Here, we will use a binary string  $y_0$  to approximate  $\bar{s}|_P$ , where  $y = y_0$  which will be defined later is a feasible solution to the following program (6). Replacing  $h_{l_i}$  by  $h_{l'_i}$ , and  $\bar{s}|_Q$  by  $h_{i_1}|_Q$ , we rewrite (1) as

$$\begin{cases} \min & d \\ d^P(y, \bar{h}_p) \leq & d - d^Q(h_{i_1}, \bar{h}_p) - d^{\bar{U}}(s, h_p); \\ d^P(y, h_{l'_i}) \leq & d - d^Q(h_{i_1}, h_{l'_i}) - d^{\bar{U}}(h_{i,2}, h_{l'_i}), i \in [1..k], \end{cases} \quad (6)$$

where  $y$  is a binary string of length  $|P|$ .

**Lemma 7.** We can obtain a solution  $y = y_0$  for (6) with objective value no more than  $(1 + \frac{1}{2^{r-1}} + 3\delta r)d_p$  in polynomial time for any fixed  $\delta > 0$ .

**Proof.** Recall that  $s^*|_P = \bar{s}|_P$  and  $s^*|_Q = h_{i_1}|_Q$ . By the definition of  $h_{i_1}$  and Lemma 1, we have

$$\begin{aligned} d^P(\bar{s}, \bar{h}_p) &= d^P(s^*, \bar{h}_p) = d^U(s^*, \bar{h}_p) - d^Q(h_{i_1}, \bar{h}_p) \\ &\leq d^U(\bar{s}, \bar{h}_p) + \frac{1}{2^{r-1}}d_p - d^Q(h_{i_1}, \bar{h}_p) \\ &\leq (1 + \frac{1}{2^{r-1}})d_p - d^Q(h_{i_1}, \bar{h}_p) - d^{\bar{U}}(s, h_p), \end{aligned}$$

where the last inequality follows from the fact  $d^U(\bar{s}, \bar{h}_p) + d^{\bar{U}}(s, h_p) \leq d_p$ . By Corollary 6, for each  $i \in [1..k]$ , we have

$$\begin{aligned} d^P(\bar{s}, h_{l'_i}) &= d^P(s^*, h_{l'_i}) = d^U(s^*, h_{l'_i}) - d^Q(h_{i_1}, h_{l'_i}) \\ &\leq (1 + \frac{1}{2^{r-1}})d_p + 2\delta|P| - d^{\bar{U}}(h_{i,2}, h_{l'_i}) - d^Q(h_{i_1}, h_{l'_i}) \\ &\leq (1 + \frac{1}{2^{r-1}} + 2\delta r)d_p - d^Q(h_{i_1}, h_{l'_i}) - d^{\bar{U}}(h_{i,2}, h_{l'_i}), \end{aligned}$$

where the last inequality follows from Lemma 2. Therefore, (6) has a solution  $y = \bar{s}|_P$  with cost  $d \leq (1 + \frac{1}{2^{r-1}} + 2\delta r)d_p$ . Let  $d_0$  be the optimal value for (6), then

$$d_0 \leq (1 + \frac{1}{2^{r-1}} + 2\delta r)d_p. \quad (7)$$

Without loss of generality, we assume that  $P = [1..|P|]$ . We use a 0-1 variable  $y_{q,a}$  to indicate whether  $y[q] = a$ , where  $q \in [1..|P|]$  and  $a \in \{0, 1\}$ . Denote  $\chi(h_{l'_i}[q], a) = 0$  if  $h_{l'_i}[q] = a$  and 1 if  $h_{l'_i}[q] \neq a$  for  $i \in [1..k]$ . Similarly, we define the constant  $\chi(\bar{h}_p[q], a)$ . Hence, the program (6) is equivalent to the following 0-1 program:

$$\left\{ \begin{array}{l} \min d \\ y_{q,0} + y_{q,1} = 1, q \in [1..|P|]; \\ \sum_{q=1}^{|P|} \sum_{a \in \{0,1\}} \chi(\bar{h}_p[q], a) y_{q,a} \leq d - d^Q(h_{i_1}, \bar{h}_p) - d^{\bar{U}}(s, h_p); \\ \sum_{q=1}^{|P|} \sum_{a \in \{0,1\}} \chi(h_{l'_i}[q], a) y_{q,a} \leq d - d^Q(h_{i_1}, h_{l'_i}) - d^{\bar{U}}(h_{i,2}, h_{l'_i}), i \in [1..k]. \end{array} \right. \quad (8)$$

Let  $y_{q,a} = \bar{y}_{q,a}$  ( $q \in [1..|P|], a \in \{0, 1\}$ ) be its fraction optimal solution with cost  $\bar{d} \leq d_0$ , which can be computed by using the method in [5]. For each  $q \in [1..|P|]$ , independently, with probability  $\bar{y}_{q,a}$ , set  $y'_{q,a} = 1$  and  $y'_{q,1-a} = 0$ , where  $a \in \{0, 1\}$ . Then, we obtain a feasible solution  $y = y'$ , where  $y'[q] = a$  if and only if  $y'_{q,a} = 1$ , for  $q \in [1..|P|]$  and  $a \in \{0, 1\}$ .

Clearly,  $d^P(y', \bar{h}_p) = \sum_{q=1}^{|P|} \sum_{a \in \{0,1\}} \chi(\bar{h}_p[q], a) y'_{q,a}$  is a sum of  $|P|$  independent 0-1 random variables. Hence,

$$\begin{aligned} E[d^P(y', \bar{h}_p)] &= \sum_{q=1}^{|P|} \sum_{a \in \{0,1\}} \chi(\bar{h}_p[q], a) E[y'_{q,a}] \\ &= \sum_{q=1}^{|P|} \sum_{a \in \{0,1\}} \chi(\bar{h}_p[q], a) \bar{y}_{q,a} \\ &\leq \bar{d} - d^Q(h_{i_1}, \bar{h}_p) - d^{\bar{U}}(s, h_p) \\ &\leq d_0 - d^Q(h_{i_1}, \bar{h}_p) - d^{\bar{U}}(s, h_p). \end{aligned}$$

Therefore, for any fixed  $\delta > 0$ , by Lemma 3, we have

$$\begin{aligned} & \mathbf{Pr}(d^P(y', \bar{h}_p) \geq d_0 - d^Q(h_{i_1}, \bar{h}_p) - d^{\bar{U}}(s, h_p) + \delta|P|) \\ & \leq \mathbf{Pr}(d^P(y', \bar{h}_p) \geq E[d^P(y', \bar{h}_p)] + \delta|P|) \\ & \leq \exp(-\frac{1}{3}\delta^2|P|). \end{aligned}$$

For each  $i \in [1..k]$ , similarly, we have

$$\mathbf{Pr}(d^P(y', h_{l'_i}) \geq d_0 - d^Q(h_{i_1}, h_{l'_i}) - d^{\bar{U}}(h_{i,2}, h_{l'_i}) + \delta|P|) \leq \exp(-\frac{1}{3}\delta^2|P|).$$

Considering all the  $k+1$  haplotypes  $\bar{h}_p, h_{l'_1}, \dots, h_{l'_k}$ , we have

$$\begin{aligned} & \mathbf{Pr}(d^P(y', \bar{h}_p) \geq d_0 - d^Q(h_{i_1}, \bar{h}_p) - d^{\bar{U}}(s, h_p) + \delta|P|, \\ & \text{or } d^P(y', s_i|_P) \geq d_0 - d^Q(h_{i_1}, h_{l'_i}) - d^{\bar{U}}(h_{i,2}, h_{l'_i}) + \delta|P|, \exists i \in [1..k]) \\ & \leq (k+1) \exp(-\frac{1}{3}\delta^2|P|). \end{aligned}$$

If  $|P| \geq 4/\delta^2 \ln(k+1)$ , then  $(k+1) \exp(-\frac{1}{3}\delta^2|P|) \leq (k+1)^{-1/3}$ . Thus, with probability at least  $1 - (k+1)^{-1/3}$ , we obtain a randomized algorithm to find a solution  $y'$  for (6) with objective value at most  $d_0 + \delta|P|$ . Since the above randomized algorithm can be derandomized by the standard method [7,12], we can find a solution  $y_0$  for (6) with objective value at most  $d_0 + \delta|P|$  in polynomial time.

If  $|P| < 4/\delta^2 \ln(k+1)$ ,  $2^{|P|} < (k+1)^{4/\delta^2}$  is polynomial in  $k+1$ . Thus, we can enumerate all possible binary strings of length  $|P|$  to find an optimal solution  $y_0$  with objective value at most  $d_0$  for (6) in polynomial time.

Thus, in both cases, we can obtain a solution  $y = y_0$  for optimization problem (6) with cost at most  $d_0 + \delta|P|$  in polynomial time. By (7) and Lemma 2, we have  $d_0 + \delta|P| \leq (1 + \frac{1}{2^{r-1}} + 3\delta r)d_p$ . This proves the lemma.

For any constant  $\epsilon \in (0, 1)$ , set  $r = \lceil \log \frac{4}{\epsilon} \rceil$  and  $\delta = \epsilon/(6r)$ . Then, we can obtain a solution  $s'$  approximating  $s|_U$  with objective value no more than  $(1+\epsilon)d_p$  for (1), where  $s'$  satisfies that  $s'|_Q = \bar{h}_{i_1}|_Q$  and  $s'|_P = \bar{y}_0$ . Hence, we obtain the following theorem.

**Theorem 8.** There is a randomized algorithm to find a ratio  $(1 + \epsilon)$  solution for (1).

Now we describe the complete randomized algorithm (randomized form) as follows.

### Algorithms for the genotype set $D$

**Input:** a genotype set  $D = \{g_1, g_2, \dots, g_k\}$ , a haplotype set  $H = \{h_1, h_2, \dots, h_m\}$ , a center index  $p$ , and a desired accuracy  $\epsilon \in (0, 1)$ .

**Output:** a haplotype  $s'$ , and a haplotype  $h_{l'_i} \in H$  for each  $i \in [1..k]$ .

**Step 1.** Compute  $s|_{\bar{U}}$  and  $h_{i,2}|_{\bar{U}}$  for each  $i \in [1..k]$ , and then construct the program (1) as in Section 2. Let  $r = \lceil \log \frac{4}{\epsilon} \rceil$  and  $\delta = \epsilon/(6r)$ .

**Step 2.** For each  $r$ -element multi-set  $\{h_{i_1}, h_{i_2}, \dots, h_{i_r}\}$ , where  $h_{i_1} = \bar{h}_p$  and  $h_{i_2}, \dots, h_{i_r} \in H$ , do

$$(a) Q = \{q \in U | h_{i_1}[q] = h_{i_2}[q] = \dots = h_{i_r}[q]\}, P = U \setminus Q.$$

(b) Let  $R$  be a multiset containing  $\lceil 4/\delta^2 \ln(mk) \rceil$  uniformly random positions from  $P$ .

(c) for every binary string  $x$  of length  $|R|$  do

(i) For each  $i \in [1..k]$ , we select the haplotype  $h_{l'_i} \in H$  minimizing

$$f(h_{l'_i}) = d^R(x, h_{l'_i}) \cdot |P|/|R| + d^Q(h_{i_1}, h_{l'_i}) + d^{\bar{U}}(h_{i,2}, h_{l'_i}).$$

(ii) Use the method given in the proof of Lemma 7 to get a solution  $y = y_0$  for (6).

(iii) Let  $s'$  be the haplotype such that  $s'|_{\bar{U}} = s|_{\bar{U}}$ ,  $s'|_Q = \bar{h}_{i_1}|_Q$  and  $s'|_P = \bar{y}_0$ . Let

$$d = \max\{d(s', h_p), d^U(\bar{s}', h_{l'_i}) + d^{\bar{U}}(h_{i,2}, h_{l'_i}), i \in [1..k]\}$$

**Step 3.** Output  $s'$  with minimum  $d$  in Step 2 and its corresponding haplotypes  $h_{l'_i} \in H$  for each  $i \in [1..k]$ .

### 3.4 Derandomization

The randomized algorithm can be derandomized using some existing techniques in [7]. The random sampling method in Section 3.2 can be derandomized as follows: Instead of randomly and independently choosing  $O(\log(mk))$  positions from  $P$ , we can pick the vertices encountered on a randomwalk of length  $O(\log(mk))$  on a constant degree expander [3]. Obviously, the number of such random walks on a constant degree expander is polynomial in terms of  $nm$ . Thus, by enumerating all random walks of length  $O(\log(mk))$ , we have a polynomial time deterministic algorithm. The randomized rounding approach in Section 3.3 can be derandomized by a method in [7].

**Theorem 9.** There is a polynomial time approximation scheme for the problem formulated in (1).

## 4 An Ultimate PTAS

In this section, we will consider the haplotypes in  $N$ . For each  $g_i \in N$ , let  $U_i$  (respectively,  $\bar{U}_i$ ) denote the set of undecided (respectively, decided) positions of  $g_i$ . By definition,  $g_i[q] = 2$  if and only if  $q \in \bar{U}_i$ . We call that  $g_i$  is *dead* if (1)  $|\bar{U} \setminus \bar{U}_i| \leq 1$  and (2) at every position  $q \in \bar{U} \cap \bar{U}_i$ , the center letter is the same as the letter of  $g_i$ . As in [9], we assume that no string  $g_i \in N$  is dead to assure the existence of a feasible solution.

**Lemma 10.** [9] Computing a haplotype pair  $(h_{i,1}, h_{i,2})$  for each  $g_i \in N$  with objective value no more than  $d_p + 1$  can be done within  $O(nLm^2)$  time.

If  $d_p \leq \frac{1}{\epsilon}$ , it is easy to verify that the optimal solution can be computed by trying all possibilities in polynomial time. Otherwise, we have  $d_p + 1 < (1 + \epsilon)d_p$ , which is the desired result.

Combining Theorem 9 in Section 3, we obtain

**Theorem 11.** There is a polynomial time approximation scheme for the closest shared center problem.

**Acknowledgements.** The work is fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project CityU 122511].

## References

1. Abecasis, G., Cherny, S., Cookson, W., Cardon, L.: Merlin-rapid analysis of dense genetic maps using sparse gene flow trees. *Nature Genetics* 30, 97–101 (2002)
2. Cai, Z., Sabaa, H., Wang, Y., Goebel, R., Wang, Z., Xu, J., Stothard, P., Lin, G.: Most parsimonious haplotype allele sharing determination. *BMC Bioinformatics* 10, 115 (2009)
3. Gillman, D.: A Chernoff bound for randomwalks on expanders. In: Proceedings of the 34th Annual Symposium on Foundations of Computer Science, pp. 680–691 (1993)
4. Gudbjartsson, D.F., Jonasson, K., Frigge, M.L., Kong, A.: Allegro, a new computer program for multipoint linkage analysis. *Nature Genetics* 25, 12–13 (2000)
5. Karmarkar, N.: A new polynomial-time algorithm for linear programming. *Combinatorica* 4, 373–395 (1984)
6. Kruglyak, L., Daly, M.J., Reeve-Daly, M.P., Lander, E.S.: Parametric and non-parametric linkage analysis: a unified multipoint approach. *American Journal of Human Genetics* 58, 1347–1363 (1995)
7. Li, M., Ma, B., Wang, L.: On the closest string and substring problems. *J. Assoc. Comput. Mach.* 49, 157–171 (2002)
8. Lin, G., Wang, Z., Wang, L., Lau, Y.-L., Yang, W.: Identification of linked regions using high-density SNP genotype data in linkage analysis. *Bioinformatics* 24(1), 86–93 (2008)
9. Ma, W., Yang, Y., Chen, Z., Wang, L.: Mutation region detection for closely related individuals without a known pedigree using high-density genotype data. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 9(2), 372–384 (2012)
10. Cui, W., Wang, L.: Identifying mutation regions for closely related individuals without a known pedigree. *BMC Bioinformatics* 13, 146 (2012)
11. Chen, Z.-Z., Ma, W., Wang, L.: The Parameterized Complexity of the Shared Center Problem, Algorithmic (to appear)
12. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge Univ. Press (1995)

# An Improved Approximation Algorithm for Scaffold Filling to Maximize the Common Adjacencies

Nan Liu<sup>1,2</sup>, Haitao Jiang<sup>1,3</sup>, Daming Zhu<sup>1</sup>, and Binhai Zhu<sup>4</sup>

<sup>1</sup> School of Computer Science and Technology, Shandong University, Jinan, China

htjiang@mail.sdu.edu.cn, dmzhu@sdu.edu.cn

<sup>2</sup> School of Computer Science and Technology, Shandong Jianzhu University, Jinan, China

liunansdu@gmail.com

<sup>3</sup> School of Mathematics and System Science, Shandong University, Jinan, China

<sup>4</sup> Department of Computer Science, Montana State University, Bozeman,

MT 59717-3880, USA

bhz@cs.montana.edu

**Abstract.** Scaffold filling is a new combinatorial optimization problem in genome sequencing. The one-sided scaffold filling problem can be described as: given an incomplete genome  $I$  and a complete (reference) genome  $G$ , fill the missing genes into  $I$  such that the number of common (string) adjacencies between the resulting genome  $I'$  and  $G$  is maximized. This problem is NP-complete for genome with duplicated genes and the best known approximation factor is 1.33, which uses a greedy strategy. In this paper, we prove a better lower bound of the optimal solution, and devise a new algorithm by exploiting the maximum matching method and a local improvement technique, which improves the approximation factor to 1.25.

## 1 Introduction

**Motivation.** The Next Generation Sequencing technology greatly improves the speed of genome sequencing, and more organisms for genome analysis can be sequenced. However, these sequences are often only a part of the complete genome. The whole genome sequencing problem is, in general, still an intractable problem. Currently, most sequencing results for genomes usually are in the form of scaffolds or contigs. Sometimes, applying these incomplete genomes for genomic analysis will introduce unnecessary errors. So it is natural to fill the missing gene fragments into the incomplete genome in a combinatorial way, and to obtain an ‘augmented’ genome which is closer to some reference genome.

**Related Results.** Muñoz *et al.* first investigated the one-sided permutation scaffold filling problem, and proposed an exact algorithm to minimize the genome rearrangement (DCJ) distance [12]. Subsequently, Jiang *et al.* considered the permutation scaffold filling under the breakpoint distance and showed that even the two-sided problem is polynomially solvable.

When genomes contain some duplicated genes, the scenario is completely different. There are three general criteria (or distance) to measure the similarity of genomes: the exemplar genomic distance [13], the minimum common string partition (MCSP)

distance [3] and the maximum number of common string adjacencies [1,10,11]. Unfortunately, unless P=NP, there does not exist any polynomial time approximation (regardless of the factor) for computing the exemplar genomic distance even when each gene is allowed to repeat three times [6,4] or even two times [2,8]. The MCSP problem is NP-complete even if each gene repeats at most two times [7] and the best known approximation factor for the general problem is  $O(\log n \log^* n)$  [3]. Based on the maximum number of common string adjacencies, Jiang *et al.* proved that the one-sided scaffold filling problem is also NP-complete, and designed a 1.33-approximation algorithm with a greedy strategy [10,11].

**Our Contribution.** In this paper, we design an approximation algorithm with a factor of 1.25 for the problem of one-sided scaffold filling to maximize the number of string adjacencies, by using a combined maximum matching and local improvement method.

## 2 Preliminaries

At first, we review some necessary definitions, which are also defined in [11]. Throughout this paper, all genes and genomes are unsigned, and it is straightforward to generalize the result to signed genomes. Given a gene set  $\Sigma$ , a string  $P$  is called *permutation* if each element in  $\Sigma$  appears exactly once in  $P$ . We use  $c(P)$  to denote the set of elements in permutation  $P$ . A string  $A$  is called *sequence* if some genes appear more than once in  $A$ , and  $c(A)$  denotes genes of  $A$ , which is a multi-set of elements in  $\Sigma$ . For example,  $\Sigma = \{a, b, c, d\}$ ,  $A = abcdacd$ ,  $c(A) = \{a, a, b, c, c, d, d\}$ . A *scaffold* is an incomplete *sequence*, typically obtained by some sequencing and assembling process. A substring with  $m$  genes (in a sequence) is called an *m-substring*, and a 2-substring is also called a *pair*, as the genes are unsigned, the relative order of the two genes of a pair does not matter, i.e., the pair  $xy$  is equal to the pair  $yx$ . Given a scaffold  $A=a_1a_2a_3\cdots a_n$ , let  $P_A = \{a_1a_2, a_2a_3, \dots, a_{n-1}a_n\}$  be the set of pairs in  $A$ .

**Definition 1.** Given two scaffolds  $A=a_1a_2\cdots a_n$  and  $B=b_1b_2\cdots b_m$ , if  $a_i a_{i+1} = b_j b_{j+1}$  (or  $a_i a_{i+1} = b_{j+1} b_j$ ), where  $a_i a_{i+1} \in P_A$  and  $b_j b_{j+1} \in P_B$ , we say that  $a_i a_{i+1}$  and  $b_j b_{j+1}$  are matched to each other. In a maximum matching of pairs in  $P_A$  and  $P_B$ , a matched pair is called an **adjacency**, and an unmatched pair is called a **breakpoint** in  $A$  and  $B$  respectively.

It follows from the definition that scaffolds  $A$  and  $B$  contain the same set of adjacencies but distinct breakpoints. The maximum matched pairs in  $B$  (or equally, in  $A$ ) form the *adjacency set* between  $A$  and  $B$ , denoted as  $a(A, B)$ . We use  $b_A(A, B)$  and  $b_B(A, B)$  to denote the set of breakpoints in  $A$  and  $B$  respectively. A gene is called a *bp-gene*, if it appears in a breakpoint. A maximal substring  $T$  of  $A$  (or  $B$ ) is call a *bp-string*, if each pair in it is a breakpoint. The leftmost and rightmost genes of a bp-string  $T$  are call the *end-genes* of  $T$ , the other genes in  $T$  are called the *mid-genes* of  $T$ . We illustrate the above definitions with the following example. Given scaffolds  $A = \langle a b c e d a b a \rangle$ ,  $B = \langle c b a b d a \rangle$ ,  $P_A = \{ab, bc, ce, ed, da, ab, ba\}$  and  $P_B = \{cb, ba, ab, bd, da\}$ . The matched pairs are  $(ab \leftrightarrow ba)$ ,  $(bc \leftrightarrow cb)$ ,  $(da \leftrightarrow da)$ ,  $(ab \leftrightarrow ab)$ .  $a(A, B) = \{ab, bc, da, ab\}$ ,  $b_A(A, B) = \{ce, ed, ba\}$ ,  $b_B(A, B) = \{bd\}$ . The bp-strings in  $A$  are *ced* and *ba*, and the bp-string in  $B$  is *bd*.

Given two scaffolds  $A=a_1a_2\cdots a_n$  and  $B=b_1b_2\cdots b_m$ , as we can see, each gene except the four ending ones is involved in two adjacencies or two breakpoints or one adjacency and one breakpoint. To get rid of this imbalance, we add “#” to both ends of  $A$  and  $B$ , which fixes a small bug in [10,11]. From now on, we assume that  $A=a_0a_1\cdots a_na_{n+1}$  and  $B=b_0b_1\cdots b_mb_{m+1}$ , where  $a_0=a_{n+1}=b_0=b_{m+1}=\#$ .

For a sequence  $A$  and a multi-set of elements  $X$ , let  $A + X$  be the set of all possible resulting sequences after filling all the elements in  $X$  into  $A$ . Now, we define the problems we study in this paper formally.

**Definition 2.** *Scaffold Filling to Maximize the Number of (String) Adjacencies (SF-MNSA).*

**Input:** two scaffolds  $A$  and  $B$  over a gene set  $\Sigma$  and two multi-sets of elements  $X$  and  $Y$ , where  $X = c(B) - c(A)$  and  $Y = c(A) - c(B)$ .

**Question:** Find  $A^* \in A + X$  and  $B^* \in B + Y$  such that  $|a(A^*, B^*)|$  is maximized.

The one-sided SF-MNSA problem is a special instance of the SF-MNSA problem where one of  $X$  and  $Y$  is empty.

**Definition 3.** *One-sided SF-MNSA.*

**Input:** a complete sequence  $G$  and an incomplete scaffold  $I$  over a gene set  $\Sigma$ , a multi-set  $X = c(G) - c(I) \neq \emptyset$  with  $c(I) - c(G) = \emptyset$ .

**Question:** Find  $I^* \in I + X$  such that  $|a(I^*, G)|$  is maximized.

Note that while the two-sided SF-MNSA problem is more general and more difficult, the One-Sided SF-MNSA problem is more practical as a lot of genome analysis are based on some reference genome [12].

We now list a few basic properties of this problem.

**Lemma 1.** *Let  $G$  and  $I$  be the input of an instance of the One-sided SF-MNSA problem, and  $x$  be any gene which appears the same times in  $G$  and  $I$ . If  $x$  does not constitute breakpoint in  $G$  (resp.  $I$ ), then it also does not constitute any breakpoint in  $I$  (resp.  $G$ ).*

**Lemma 2.** *Let  $G$  and  $I$  be the input of an instance of the One-sided SF-MNSA problem, let  $bp(I)$  and  $bp(G)$  be the multi-set of bp-genes in  $I$  and  $G$  respectively. Then any gene in  $bp(G)$  appears in  $bp(I) \cup X$ , and  $bp(I) \subseteq bp(G)$ .*

*Proof.* Assume to the contrary that there exists a gene  $x$ ,  $x \in bp(G)$ , but  $x \notin bp(I) \cup X$ . Since  $x \notin X$ ,  $x$  appears the same number of times in  $G$  and  $I$ ; moreover,  $x \notin bp(I)$ , then all the pairs in  $I$  containing  $x$  are adjacencies. From Lemma 1, all the pairs involving  $x$  in  $G$  are adjacencies, contradicting the assumption that  $x \in bp(G)$ . So any gene in  $bp(G)$  appears in  $bp(I) \cup X$ . By a similar argument, we can prove  $bp(I) \subseteq bp(G)$ .  $\square$

Each breakpoint contains two genes, from what we discussed in Lemma 2, every breakpoint in the complete sequence  $G$  belongs to one of the three multi-sets according to the affiliation of its two bp-genes.

- $BP_1(G)$ : breakpoints with one bp-gene in  $X$  and the other bp-gene not in  $X$ .
- $BP_2(G)$ : breakpoints with both of the bp-genes in  $X$ .
- $BP_3(G)$ : breakpoints with both of the bp-genes not in  $X$ .

### 3 Approximation Algorithm for One-Sided SF-MNSA

In this section, we present a 1.25-Approximation algorithm for the one-sided SF-MNSA problem. The goal of solving this problem is, while inserting the genes of  $X$  into the scaffold  $I$ , to obtain as many adjacencies as possible. No matter in what order the genes are inserted, they appear in groups in the final  $I' \in I + X$ , so we can consider that  $I'$  is obtained by inserting strings (composed of genes of  $X$ ) into  $I$ .

Obviously, inserting a string of length one (i.e., a single gene) will generate at most two adjacencies, and inserting a string of length  $m$  will generate at most  $m+1$  adjacencies. Therefore, we will have two types of inserted strings.

1. Type-1: a string of  $k$  missing genes  $x_1, x_2, \dots, x_k$  are inserted in between  $y_i y_{i+1}$  in the scaffold  $I$  to obtain  $k+1$  adjacencies (i.e.,  $y_i x_1, x_1 x_2, \dots, x_{k-1} x_k, x_k y_{i+1}$ ), where  $y_i y_{i+1}$  is a breakpoint.  
In this case,  $x_1 x_2 \dots x_k$  is called a  $k$ -Type-1 string,  $y_i y_{i+1}$  is called a *dock*, and we also say that  $y_i y_{i+1}$  docks the corresponding  $k$ -Type-1 string  $x_1 x_2 \dots x_k$ .
2. Type-2: a sequence of  $l$  missing genes  $z_1, z_2, \dots, z_l$  are inserted in between  $y_j y_{j+1}$  in the scaffold  $I$  to obtain  $l$  adjacencies (i.e.,  $y_j z_1$  or  $z_l y_{j+1}, z_1 z_2, \dots, z_{l-1} z_l$ ), where  $y_j y_{j+1}$  is a breakpoint; or a sequence of  $l$  missing genes  $z_1, z_2, \dots, z_l$  are inserted in between  $y_j y_{j+1}$  in the scaffold  $I$  to obtain  $l+1$  adjacencies (i.e.,  $y_j z_1, z_1 z_2, \dots, z_{l-1} z_l, z_l y_{j+1}$ ), where  $y_j y_{j+1}$  is an adjacency.

This is the basic observation for devising our algorithm. Most of our work is devoted to searching the Type-1 strings.

#### 3.1 Searching the 1-Type-1 Strings

To identify the 1-Type-1 strings, we construct a bipartite graph  $\mathcal{G}_1 = (X, b_I(I, G), E)$ , where for each gene  $x_i$  of  $X$  and each breakpoint  $y_j y_{j+1}$  of  $b_I(I, G)$ , if we can obtain two adjacencies by inserting  $x_i$  in between  $y_j y_{j+1}$ , then there is an edge connecting  $x_i$  to  $y_j y_{j+1}$ . Therefore, a matching of  $\mathcal{G}_1$  gives us the 1-Type-1 strings and their corresponding docks. In our algorithm, we compute a maximum matching of  $\mathcal{G}_1$ , then the number of 1-Type-1 strings obtained by our algorithm will not be smaller than that of the optimal solution.

Algorithm 1: *Max-Matching*( $G, I$ )

- ```

1 Construct a bipartite graph  $\mathcal{G}_1 = (X, b_I(I, G), E)$ ,
    $(x_i, y_j y_{j+1}) \in E$  iff  $y_j y_{j+1}$  docks  $x_i$ , for all  $x_i \in X$  and  $y_j y_{j+1} \in b_I(I, G)$ .
2 Compute a maximum matching  $M$  of  $\mathcal{G}_1$ .
3 Return  $M$ .

```

#### 3.2 Searching the 2-Type-1 Strings

To identify the 2-Type-1 strings, we first find a set  $Q$  of 2-Type-1 strings greedily. For  $Q$ , let  $c(Q)$  be the multi-set of genes in  $Q$ , and let  $D(Q)$  be the set of corresponding docks. Then, we improve  $Q$  to find more 2-Type-1 strings using a local search method. There are two ways to improve  $Q$ :

1. Delete a 2-Type-1 string  $x_i x_j$  from  $Q$ , release its corresponding dock  $y_p y_{p+1}$ , construct two new 2-Type-1 strings  $x_i x_k$  and  $x_j x_l$ , where  $x_k, x_l \notin c(Q)$ .
2. Delete a 2-Type-1 string  $x_i x_j$  from  $Q$ , release its corresponding dock  $y_p y_{p+1}$ , construct two new 2-Type-1 strings  $x_i x_k$  and  $x_r x_l$ , where  $x_r x_l$  docks  $y_p y_{p+1}$ , and  $x_k, x_l, x_r \notin c(Q)$ .

Algorithm 2: *Local-Optimize*( $G, I$ )

```

1 Identify the breakpoints and adjacencies in  $G$  and  $I$ ,  $Q \leftarrow \emptyset$ ,  $D(Q) \leftarrow \emptyset$ .
2 WHILE ( $x_i x_j$  is docked at  $y_p y_{p+1}$ , where  $x_i, x_j \in X$  and  $y_p y_{p+1} \in b_I(I, G)$ )
3   {  $Q \leftarrow Q \cup \{x_i x_j\}$ ,  $D(x_i x_j) \leftarrow \{y_p y_{p+1}\}$ ,
     $X \leftarrow X - \{x_i, x_j\}$ ,  $b_I(I, G) \leftarrow b_I(I, G) - \{y_p y_{p+1}\}$  }.
4 For each 2-Type-1 string  $x_i x_j \in Q$  docked at  $y_p y_{p+1} \in b_I(I, G)$ 
  4.1 if ( $x_i x_k$  is docked at  $y_u y_{u+1}$ ,  $x_j x_l$  is docked at  $y_v y_{v+1}$ , where  $x_k, x_l \in X$ ,
         $y_u y_{u+1}, y_v y_{v+1} \in b_I(I, G)$ )
    then {  $Q \leftarrow Q \cup \{x_i x_k, x_j x_l\} - \{x_i x_j\}$ ,  $D(x_i x_k) \leftarrow \{y_u y_{u+1}\}$ ,
            $D(x_j x_l) \leftarrow \{y_v y_{v+1}\}$ ,  $X \leftarrow X - \{x_k, x_l\}$ ,
            $b_I(I, G) \leftarrow b_I(I, G) \cup \{y_p y_{p+1}\} - \{y_u y_{u+1}, y_v y_{v+1}\}$  }.
    goto step 4.
  4.2 if ( $x_i x_k$  is docked at  $y_u y_{u+1}$ ,  $x_l x_r$  is docked at  $y_p y_{p+1}$ , where  $x_k, x_l, x_r \in X$ ,
         $y_u y_{u+1} \in b_I(I, G)$ )
    then {  $Q \leftarrow Q \cup \{x_i x_k, x_l x_r\} - \{x_i x_j\}$ ,  $D(x_i x_k) \leftarrow \{y_u y_{u+1}\}$ ,
            $D(x_l x_r) \leftarrow \{y_p y_{p+1}\}$ ,  $X \leftarrow X - \{x_k, x_l, x_r\} \cup \{x_j\}$ ,
            $b_I(I, G) \leftarrow b_I(I, G) - \{y_u y_{u+1}\}$  }.
    goto step 4.
5 Return  $Q$  and  $D(Q)$ .

```

### 3.3 Searching the 3-Type-1 Strings

To search the 3-Type-1 strings, we use a greedy method.

Algorithm 3: *Greedy-Search*( $G, I$ )

```

1 Identify the breakpoints and adjacencies in  $G$  and  $I$ ,  $F \leftarrow \emptyset$ ,  $D(F) \leftarrow \emptyset$ .
2 WHILE ( $x_i x_j x_k$  is docked at  $y_w y_{w+1}$ , where  $x_i, x_j, x_k \in X$  and  $y_w y_{w+1} \in b_I(I, G)$ )
3   {  $F \leftarrow F \cup \{x_i x_j x_k\}$ ,  $D(x_i x_j x_k) \leftarrow \{y_w y_{w+1}\}$ ,
      $X \leftarrow X - \{x_i, x_j, x_k\}$ ,  $b_I(I, G) \leftarrow b_I(I, G) - \{y_p y_{p+1}\}$  }.
4 Return  $F$  and  $D(F)$ .

```

### 3.4 Inserting the Remaining Genes

In this subsection, we present a polynomial time algorithm guaranteeing that the number of adjacencies increases by the same number of the genes inserted. A general idea of this algorithm was mentioned in [11], with many details missing, and we will present the details here.

Given the complete sequence  $G$  and the scaffold  $I$ , as we discussed in Section 2, the breakpoints in  $G$  can be divided into three sets:  $BP_1(G)$ ,  $BP_2(G)$ , and  $BP_3(G)$ . In any case, the breakpoints in  $BP_3(G)$  cannot be converted into adjacencies; so we try to convert the breakpoints in  $BP_1(G)$  and  $BP_2(G)$  into adjacencies.

**Lemma 3.** *If  $BP_1(G) \neq \emptyset$ , then there exists a breakpoint in  $I$  where after some gene of  $X$  is inserted, the number of adjacencies increases by one.*

*Proof.* Let  $t_i t_{i+1}$  be a breakpoint in  $G$ , satisfying that  $t_i t_{i+1} \in BP_1(G)$ ,  $t_i \in X$ , and, from Lemma 2,  $t_{i+1} \in bp(I)$ . Then, there exists a breakpoint  $t_{i+1} s_j$  or  $s_k t_{i+1}$  in  $I$ . Hence, if we insert  $t_i$  in between that breakpoint, we will obtain a new adjacency  $t_i t_{i+1}$  without affecting any other adjacency.  $\square$

Thus, it is trivial to obtain one more adjacency whenever  $BP_1(G) \neq \emptyset$ .

**Lemma 4.** *For any  $x \in X \cap c(I)$ , if there is an “ $xx$ ” breakpoint in  $G$  then after inserting  $x$  in between some “ $xy$ ” pair in  $I$ , the number of adjacencies increases by one.*

*Proof.* If “ $xy$ ” is a breakpoint, then after inserting an ‘ $x$ ’ in between it, we obtain a new adjacencies “ $xx$ ”. If “ $xy$ ” is an adjacency, then after inserting an ‘ $x$ ’ in between it, we have “ $xx$ ”. The adjacency “ $xy$ ” still exists, and we obtain a new adjacencies “ $xx$ ”.  $\square$

**Lemma 5.** *If there is a breakpoint “ $xy$ ” in  $BP_2(G)$  and a breakpoint “ $xz$ ” (resp. “ $yz$ ”) in  $I$ , then after inserting  $y$  (resp.  $x$ ) in between “ $xz$ ” (resp. “ $yz$ ”) in  $I$ , the number of adjacencies increases by one.*

*Proof.* From the definition of  $BP_2(G)$ , we know that  $x, y \in X$ . Since “ $xy$ ” is a breakpoint in  $G$  and “ $xz$ ” is a breakpoint in  $I$ , we obtain a new adjacency “ $xy$ ” by inserting  $y$  in between “ $xz$ ”, without affecting any other adjacency. A similar argument for inserting  $x$  in between “ $yz$ ” also holds.  $\square$

Next, we show that the following case is polynomially solvable. This case satisfies the following conditions.

1.  $BP_1(G) = \emptyset$ ;
2. It does not contain a breakpoint like “ $xx$ ” in  $G$  unless  $x \notin X \cap c(I)$ ;
3. For any breakpoint of the form “ $xy$ ” in  $BP_2(G)$ , all the pairs in  $I$  involving  $x$  or  $y$  are adjacencies.

Let  $BS_2(G)$  be the set of bp-strings in  $G$  with all breakpoints belonging to  $BP_2(G)$ .

**Lemma 6.** *In the case satisfying (1), (2) and (3), the number of times a gene appears as an end-gene of some bp-string of  $BS_2(G)$  is even.*

From Lemma 6, if we denote each bp-string of  $BS_2(G)$  by a vertex, and there is an edge between two vertices iff their corresponding bp-strings have a common end-gene, the resulting graph contains a cycle of distinct vertices. Traveling this cycle, concatenating the bp-strings corresponding to the vertices, and deleting one copy of the common end-gene, eventually we can obtain a string composed of genes of  $X$ . The following lemma and corollary shows that this string can be inserted into  $I$  entirely, generating no breakpoint at all.

**Lemma 7.** *In the case satisfying (1),(2) and (3), for a gene  $x$ , let  $q_1$  be the number that it appears as an end-gene, let  $q_2$  be the number that it appears in some bp-string of  $BS_2(G)$  as a mid-gene, and let  $r$  be the number that it appears in  $X$ . Then, we have  $r = q_1/2 + q_2$ .*

We can summarize the above ideas as the algorithm Insert-Whole-Strings( $\bullet$ ), which ensures us to obtain as many adjacencies as the number of missing genes inserted. The details will be given in the full version.

**Theorem 1.** *The algorithm Insert-Whole-Strings( $\bullet$ ) guarantees that the number of adjacencies increased is not smaller than the number of genes inserted.*

## 4 Analysis of the Approximation Algorithm

### 4.1 A Lower Bound

Given an instance of One-sided SF-MNSA, let  $I^* \in I+X$  be the final scaffold in the optimal solution after inserting all genes of  $X$  into  $I$ . Compared to  $I$ , all genes belonging to  $X$  appear as substrings in  $I^*$ . Let  $x_1x_2\dots x_l$  be a string inserted in between  $y_iy_{i+1}$  in  $I^*$ , then either  $y_ix_1$  or  $x_ly_{i+1}$  or both are adjacencies. Since otherwise, we could delete this string from  $I^*$  (number of adjacencies decreases by at most  $l-1$ ), re-insert it following the algorithm Insert-Whole-Strings( $\bullet$ ) (number of adjacencies increases by at least  $l$ ), and obtain one more adjacency. Thus, we have the following corollary of Theorem 1,

**Corollary 1.** *Each substring in  $I^*$  composed of genes of  $X$  is either Type-1 or Type-2.*

Now, we present a new lower bound for the optimal number of adjacencies.

**Lemma 8.** *Let  $OPT$  be the number of adjacencies between  $G$  and  $I^*$ ,  $k_0$  be the number of adjacencies between  $G$  and  $I$ , and  $k_1=|X|$ . Let  $b_i$  be the number of  $i$ -Type-1 substrings and  $q$  be the maximum length of Type-1 substrings in the optimal solution between  $G$  and  $I^*$ . Then*

$$OPT - k_0 = k_1 + b_1 + b_2 + \dots + b_q \leq \frac{5}{4}(k_1 + \frac{3}{5}b_1 + \frac{2}{5}b_2 + \frac{1}{5}b_3) \quad (1)$$

Following Lemma 8, if the number of Type-1 substrings computed in the approximation algorithm is not smaller than  $(3b_1 + 2b_2 + b_3)/5$ , then the approximation factor is  $5/4$ .

### 4.2 Description of the Main Algorithm

There are four steps in our algorithm. Firstly, we try to search the 1-Type-1 strings; secondly, we try to search the 2-Type-1 strings; thirdly, we try to search the 3-Type-1 strings; finally, we insert the remaining genes in  $X$ , guaranteeing that on average we will obtain at least one adjacency for each inserted missing gene.

**Main Algorithm**

**Input:** Complete sequence  $G$  and incomplete scaffold  $I$ ,  $X = c(G) - c(I)$ .

**Output:**  $I' \in I + X$

- 1 Call  $\text{Max-Matching}(G, I)$ , which returns a maximum matching  $M$ .  
 { Let  $M_1$  be the set of 1-Type-1 strings and  $D(M_1)$  be their corresponding docks.  
   Insert the 1-Type-1 strings into their corresponding docks.  
   The resulting incomplete scaffold is denoted as  $I_1$ . }
- 2 Call  $\text{Local-Optimize}(G, I_1)$ , which returns  $Q$  and  $D(Q)$ .  
 { Insert the 2-Type-1 strings of  $Q$  into their corresponding docks.  
   The resulting incomplete scaffold is denoted as  $I_2$ . }
- 3 Call  $\text{Greedy-Search}(G, I_2)$ , which returns  $F$  and  $D(F)$ .  
 { Insert the 3-Type-1 strings of  $F$  into their corresponding docks.  
   The resulting incomplete scaffold is denoted as  $I_3$ . }
- 4 Call  $\text{Insert-Whole-Strings}(G, I_3)$ .  
 { Let the resulting complete scaffold be  $I'$ . }
- 5 Return  $I'$ .

### 4.3 Proof of the Approximation Factor

In our algorithm, we make effort to insert Type-1 substrings as much as possible. But a Type-1 substring (say  $I_s$ ) inserted by our algorithm may make other Type-1 substrings in some optimal solution infeasible, we say  $I_s$  *destroys* them. The following lemma shows the number of Type-1 substrings that could be destroyed by a given Type-1 substring.

**Lemma 9.** *A  $i$ -Type-1 substring can destroy at most  $i+1$  Type-1 substrings in some optimal solution.*

*Proof.* Assume that an  $i$ -Type-1 substring  $I_s$  is inserted in between some breakpoint  $y_jy_{j+1}$  in  $I$ . Then each of the genes in  $I_s$ , if not used by  $I_s$ , could form a distinct Type-1 substring in some optimal solution. Also, there may exist another Type-1 substring that could be inserted in between the breakpoint  $y_jy_{j+1}$  in the optimal solution. Totally, at most  $i+1$  Type-1 substrings in the optimal solution could be destroyed by  $I_s$ .  $\square$

Next, we will compare the number of  $i$ -Type-1 substrings between our algorithm and some optimal solution. Above all, we focus on the 1-Type-1 substrings, which are computed by the algorithm  $\text{Max-Matching}(\bullet)$ .

**Lemma 10.** *The 1-Type-1 substrings computed in our algorithm is not smaller than those in any optimal solution.*

For the simplicity of comparison, given an edge  $e \in M$  and an edge  $\tilde{e} \in W$ , we say that  $e$  *destroys*  $\tilde{e}$  if  $e$  and  $\tilde{e}$  have exactly one common endpoint. Actually,  $e$  *destroys*  $\tilde{e}$  implies that the 1-Type-1 substring corresponding to  $e$  makes the 1-Type-1 substring corresponding to  $\tilde{e}$  *infeasible*.

Since  $M$  is a maximum matching, each connected component of  $M \cup W$  is either a path or an even cycle, and at least one end-edge of a path belongs to  $M$ . So, for each edge in  $W - M$ , we can assign a distinct edge in  $M - W$  that destroys it. Under this assignment, each edge in  $M - W$  destroys at most one edge in  $W - M$ , i.e., a 1-Type-1

substring of our algorithm can destroy at most one 1-Type-1 substring of the optimal solution.

Cases of such substrings which are destroyed by a 1-Type-1 substring of our algorithm (except the 1-Type-1 substrings of  $M \cap W$ ), are described in the following Table 1, where  $b'_{1j}$  denotes the number of 1-Type-1 substrings of the  $j$ th case and “other” means an  $i$ -Type-1 substring ( $i > 3$ ).

**Table 1.** Cases of substrings destroyed by a 1-Type-1 substring

| number of 1-Type-1 substrings | one substring destroyed | another substring destroyed |
|-------------------------------|-------------------------|-----------------------------|
| $b'_{11}$                     | 1-Type-1                | 2-Type-1                    |
| $b'_{12}$                     | 1-Type-1                | 3-Type-1                    |
| $b'_{13}$                     | 1-Type-1                | other or none               |
| $b'_{14}$                     | 2-Type-1                | 2-Type-1                    |
| $b'_{15}$                     | 2-Type-1                | 3-Type-1                    |
| $b'_{16}$                     | 2-Type-1                | other or none               |
| $b'_{17}$                     | 3-Type-1                | 3-Type-1                    |
| $b'_{18}$                     | 3-Type-1                | other or none               |
| $b'_{19}$                     | other                   | other or none               |

Let  $b'_1$  be the number of 1-Type-1 substrings computed by our algorithm, i.e.,  $b'_1 = |M|$ . Then we have,

$$b'_1 = b'_{11} + b'_{12} + b'_{13} + \dots + b'_{19} + |M \cap W| \quad (2)$$

Now, we will analyze the number of 2-Type-1 substrings, which are computed by the algorithm *Local-Optimize(•)*.

**Lemma 11.** *Let  $b_2$  be the number of 2-Type-1 substrings of some optimal solution,  $b'_2$  be the number of 2-Type-1 substrings obtained by our algorithm. Then*

$$b'_2 \geq \frac{1}{2} \times (b_2 - b'_{11} - 2b'_{14} - b'_{15} - b'_{16}). \quad (3)$$

*Proof.* Let  $Q$  be the set of 2-Type-1 substrings obtained at step 2 (by the *Local-Optimize(•)* algorithm), and  $P$  be the set of 2-Type-1 substrings in some optimal solution which does not include those 2-Type-1 substrings destroyed at step 1 (by the *Max-Matching(•)* algorithm), i.e.,  $|Q| = b'_2$ ,  $|P| = b_2 - b'_{11} - 2b'_{14} - b'_{15} - b'_{16}$ . Let  $R = Q \cap P$  be the set of 2-Type-1 substrings which are contained in both  $Q$  and  $P$ . We just proceed to compare the number of 2-Type-1 substrings in  $P - R$  and  $Q - R$ , even though we cannot compute  $P - R$  explicitly.

Let  $Q - R = \{x_1x_2, x_3x_4, \dots, x_{2q-1}x_{2q}\}$  and  $P - R = \{y_1y_2, y_3y_4, \dots, y_{2p-1}y_{2p}\}$ . For simplicity, we mark these gene appearing more than one times in  $P - R$  (and  $Q - R$ ) with distinct labels. Consider an imaginary bipartite graph  $\mathcal{G}' = (Q - R, P - R, E)$ ,

where there exists an edge between two vertices iff their corresponding 2-Type-1 substrings share a common gene (same gene and same label). Then, each vertex in  $\mathcal{G}'$  has degree at most two.

For a vertex  $x_i x_{i+1}$  of  $Q - R$  and an isolated vertex  $y_j y_{j+1}$  in  $P - R$ , if both  $x_i x_{i+1}$  and  $y_j y_{j+1}$  dock  $s_k s_{k+1}$ , then mark both  $x_i x_{i+1}$  and  $y_j y_{j+1}$  with a same color. Each such pair are marked with a distinct color. Since in the *Local-Optimize*(•) algorithm the original 2-Type-1 substrings in  $Q$  are formed by a greedy search, any isolated vertex of  $P - R$  in  $\mathcal{G}'$  must be colored. So  $\mathcal{G}'$  is composed of disjoint paths, even cycles, colored isolated vertices of  $P - R$  and isolated vertices of  $Q - R$ .

In an even cycle or even path (i.e., with an even number of vertices), the number of vertices in  $Q - R$  and  $P - R$  is equal, and in the worst case all the vertices on even cycles/paths in  $Q - R$  are colored.

For an odd path (i.e., with an odd number of vertices), as  $\mathcal{G}'$  is bipartite, the two endpoints are either both in  $P - R$  or both in  $Q - R$ . In an odd path with both endpoints in  $Q - R$ , the number of vertices in  $Q - R$  is one more than that in  $P - R$ .

Let  $p_1, q_1, p_2, q_2, \dots, p_r, q_r, p_{r+1}$  be an odd path in  $\mathcal{G}'$ , where  $p_i \in (P - R)$  ( $1 \leq i \leq r + 1$ ), and  $q_j \in (Q - R)$  ( $1 \leq j \leq r$ ). We claim that  $q_1$  cannot be colored. Since otherwise, let  $p_t$  be the isolated vertex that has the same color with  $q_1$ , according to step 4.2 of the *Local-Optimize*(•) algorithm,  $q_1$  should be deleted from  $Q$ ,  $p_1$  and  $p_t$  should be added to  $Q$ . This contradicts the local optimality of  $Q$ . A similar claim holds for  $q_t$ . So each odd path in  $\mathcal{G}'$  ends with some vertices in  $P - R$  whose neighbors are uncolored.

Let  $l$  be the number of odd paths in  $\mathcal{G}'$  which end with some vertices in  $P - R$ . Putting all together, we can conclude that: (1) the number of isolated colored vertices in  $P - R$  is at most  $|Q - R| - l$ ; (2) the vertices in  $P - R$  appearing on the cycles or paths in  $\mathcal{G}'$  is at most  $|Q - R| + l$ . Then  $|P - R| \leq 2|Q - R|$ . So  $|P| \leq 2|Q|$ . Since  $|Q| = b'_2$  and  $|P| = b_2 - b'_{11} - 2b'_{14} - b'_{15} - b'_{16}$ , we have  $b'_2 \geq (b_2 - b'_{11} - 2b'_{14} - b'_{15} - b'_{16})/2$ , hence the lemma follows.  $\square$

When computing the 2-Type-1 substrings at Step 2, except the 2-Type-1 substrings of  $R = Q \cap P$ , these 2-Type-1 substrings computed by our algorithm could destroy some other Type-1 substrings of the optimal solution. It follows from Lemma 9 that a 2-Type-1 substring can destroy at most three Type-1 substrings in some optimal solution. Cases of such substrings, which are destroyed by a 2-Type-1 substring of our algorithm, are described in the following Table 2, where  $b'_{2j}$ s denote the number of 2-Type-1 substring in the  $j$ th case and “other” means an  $i$ -Type-1 substring ( $i > 3$ ).

Let  $b'_2$  denote the number of 2-Type-1 substrings computed by our algorithm, then we have  $b'_2 = b'_{20} + b'_{21} + b'_{22} + b'_{23} + \dots + b'_{29} + |R|$ , where  $R = Q \cap P$ .

At Step 3 (i.e., the *Greedy-Search*(•) algorithm), let  $Z$  be the set of 3-Type-1 substrings both in our algorithm and in the optimal solution. Except those 3-Type-1 substrings of  $Z$ , a 3-Type-1 substring computed by our algorithm can destroy at most four Type-1 substrings in some optimal solution. Cases of such substrings which are destroyed by a 3-Type-1 substring are described in the following Table 3, where  $b'_{3j}$ s denote the number of 3-Type-1 substrings of each case  $j$  and “other” means an  $i$ -Type-1 substring ( $i > 3$ ).

**Table 2.** Cases of substrings destroyed by a 2-Type-1 substring

| number of 2-Type-1 substrings | the first substring destroyed | the second substring destroyed | the third substring destroyed |
|-------------------------------|-------------------------------|--------------------------------|-------------------------------|
| $b'_{20}$                     | 2-Type-1                      | 2-Type-1                       | 2-Type-1                      |
| $b'_{21}$                     | 2-Type-1                      | 2-Type-1                       | 3-Type-1                      |
| $b'_{22}$                     | 2-Type-1                      | 2-Type-1                       | other or none                 |
| $b'_{23}$                     | 2-Type-1                      | 3-Type-1                       | 3-Type-1                      |
| $b'_{24}$                     | 2-Type-1                      | 3-Type-1                       | other or none                 |
| $b'_{25}$                     | 2-Type-1                      | other or none                  | other or none                 |
| $b'_{26}$                     | 3-Type-1                      | 3-Type-1                       | 3-Type-1                      |
| $b'_{27}$                     | 3-Type-1                      | 3-Type-1                       | other or none                 |
| $b'_{28}$                     | 3-Type-1                      | other or none                  | other or none                 |
| $b'_{29}$                     | other                         | other or none                  | other or none                 |

Let  $b'_3$  denote the number of 3-Type-1 substrings of our algorithm, then we have  $b'_3 = b'_{31} + b'_{32} + b'_{33} + b'_{34} + b'_{35} + |Z|$ , where and  $Z$  is the set of 3-Type-1 substrings both in our algorithm and in the optimal solution.

**Table 3.** Cases of substrings destroyed by a 3-Type-1 substring

| number of 3-Type-1 substrings | the 1st substring destroyed | the 2nd substring destroyed | the 3rd substring destroyed | the 4th substring destroyed |
|-------------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
| $b'_{31}$                     | 3-Type-1                    | 3-Type-1                    | 3-Type-1                    | 3-Type-1                    |
| $b'_{32}$                     | 3-Type-1                    | 3-Type-1                    | 3-Type-1                    | other or none               |
| $b'_{33}$                     | 3-Type-1                    | 3-Type-1                    | other or none               | other or none               |
| $b'_{34}$                     | 3-Type-1                    | other or none               | other or none               | other or none               |
| $b'_{35}$                     | other                       | other or none               | other or none               | other or none               |

**Lemma 12.** Let  $b'_1, b'_2, b'_3$  be the number of new 1-Type-1, 2-Type-1 and 3-Type-1 substrings inserted at step1, step 2 and step 3 in our main algorithm respectively. Then

$$b'_1 + b'_2 + b'_3 \geq \frac{1}{5} \times (3b_1 + 2b_2 + b_3).$$

**Theorem 2.** The One-sided SF-MNSA problem admits a polynomial time factor-1.25 approximation.

*Proof.* Following the approximation algorithm, Theorem1, Lemma 8, and Lemma 12, we have the approximation solution value  $APP$ , which satisfies the following inequalities.

$$APP - k_0 = k_1 + b'_1 + b'_2 + b'_3 \geq k_1 + \frac{3}{5}b_1 + \frac{2}{5}b_2 + \frac{1}{5}b_3 \geq \frac{4}{5}(OPT - k_0).$$

So, we have  $APP \geq \frac{4}{5}OPT + \frac{1}{5}k_0 \geq \frac{4}{5}OPT$ . Hence  $\frac{OPT}{APP} \leq 1.25$ , and the theorem is proven.  $\square$

## 5 Concluding Remarks

In this paper, we used a mixture of maximum matching and local improvement methods to obtain a factor-1.25 approximation for One-sided MNSA. It would be interesting to know whether the 1.25 factor can be further improved.

**Acknowledgments.** This research is partially supported by NSF of China projects 60070019 and 60928006, by the Doctoral Fund of Ministry of Education of China under grant 20090131110009, by China Postdoctoral Science Foundation under grant 2011M501133, by NSF under grant DMS-0918034, and by the Open Fund of Top Key Discipline of Computer Software and Theory in Zhejiang Provincial Colleges at Zhejiang Normal University.

## References

1. Angibaud, S., Fertin, G., Rusu, I., Thevenin, A., Vialette, S.: On the approximability of comparing genomes with duplicates. *J. Graph Algorithms and Applications* 13(1), 19–53 (2009)
2. Blin, G., Fertin, G., Sikora, F., Vialette, S.: The EXEMPLAR BREAKPOINT DISTANCE for non-trivial genomes cannot be approximated. In: Das, S., Uehara, R. (eds.) WALCOM 2009. LNCS, vol. 5431, pp. 357–368. Springer, Heidelberg (2009)
3. Cormode, G., Muthukrishnan, S.: The string edit distance matching problem with moves. In: Proc. 13th ACM-SIAM Symp. on Discrete Algorithms (SODA 2002), pp. 667–676 (2002)
4. Chen, Z., Fowler, R., Fu, B., Zhu, B.: On the inapproximability of the exemplar conserved interval distance problem of genomes. *J. Combinatorial Optimization* 15(2), 201–221 (2008)
5. Chen, Z., Fu, B., Xu, J., Yang, B., Zhao, Z., Zhu, B.: Non-breaking similarity of genomes with gene repetitions. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 119–130. Springer, Heidelberg (2007)
6. Chen, Z., Fu, B., Zhu, B.: The approximability of the exemplar breakpoint distance problem. In: Cheng, S.-W., Poon, C.K. (eds.) AAIM 2006. LNCS, vol. 4041, pp. 291–302. Springer, Heidelberg (2006)
7. Goldstein, A., Kolman, P., Zheng, J.: Minimum common string partition problem: Hardness and approximations. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 484–495. Springer, Heidelberg (2004); also in: *The Electronic Journal of Combinatorics* 12, paper R50 (2005)
8. Jiang, M.: The zero exemplar distance problem. In: Tannier, E. (ed.) RECOMB-CG 2010. LNCS, vol. 6398, pp. 74–82. Springer, Heidelberg (2010)
9. Jiang, H., Zheng, C., Sankoff, D., Zhu, B.: Scaffold filling under the breakpoint distance. In: Tannier, E. (ed.) RECOMB-CG 2010. LNCS, vol. 6398, pp. 83–92. Springer, Heidelberg (2010)
10. Jiang, H., Zhong, F., Zhu, B.: Filling scaffolds with gene repetitions: Maximizing the number of adjacencies. In: Giancarlo, R., Manzini, G. (eds.) CPM 2011. LNCS, vol. 6661, pp. 55–64. Springer, Heidelberg (2011)
11. Jiang, H., Zheng, C., Sankoff, D., Zhu, B.: Scaffold filling under the breakpoint and related distances. *IEEE/ACM Trans. Bioinformatics and Comput. Biology* 9(4), 1220–1229 (2012)
12. Muñoz, A., Zheng, C., Zhu, Q., Albert, V., Rounseley, S., Sankoff, D.: Scaffold filling, contig fusion and gene order comparison. *BMC Bioinformatics* 11, 304 (2010)
13. Sankoff, D.: Genome rearrangement with gene families. *Bioinformatics* 15(11), 909–917 (1999)

# An Efficient Algorithm for One-Sided Block Ordering Problem with Block-Interchange Distance

Kun-Tze Chen<sup>1</sup>, Chi-Long Li<sup>1</sup>, Chung-Han Yang<sup>2</sup>, and Chin Lung Lu<sup>1,\*</sup>

<sup>1</sup> Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan  
[cLLU@cs.nthu.edu.tw](mailto:cLLU@cs.nthu.edu.tw)

<sup>2</sup> Institute of Bioinformatics and Systems Biology, National Chiao Tung University, Hsinchu, Taiwan

**Abstract.** In this work, we study the one-sided block ordering problem with block-interchange distance. Given two signed permutations  $\pi$  and  $\sigma$  of size  $n$ , where  $\pi$  represents a partially assembled genome consisting of several blocks (contigs) and  $\sigma$  represents a completely assembled genome, the one-sided block ordering problem is to order (assemble) the blocks of  $\pi$  such that the block-interchange distance between the assembly of  $\pi$  and  $\sigma$  is minimized. In addition to genome rearrangements and phylogeny reconstruction, the one-sided block ordering problem is useful in genome resequencing, because its algorithms can be used to assemble the contigs of partially assembled resequencing genomes based on their completely assembled genomes. By using permutation groups, we design an efficient algorithm to solve the one-sided block ordering problem with block-interchange distance in  $\mathcal{O}(n \log n)$  time. Moreover, we show that the assembly of  $\pi$  can be done in  $\mathcal{O}(n)$  time and its block-interchange distance from  $\sigma$  can also be calculated in advance in  $\mathcal{O}(n)$  time.

## 1 Introduction

The next-generation DNA sequencing techniques have greatly advanced in the past decade [1,2], allowing an increasing number of draft genomes to be produced rapidly in a decreasing cost. Usually, these draft genomes are partially sequenced, leading to their published genomes as collections of unassembled contigs. However, these draft genomes in contig form cannot be used immediately by current genome rearrangement algorithms, because these rearrangement algorithms need the completely assembled genomes to calculate their genome rearrangement distances, such as reversal, transposition, block-interchange and/or translocation distances, which are further required for the studies of rearrangement-based phylogeny [3,4]. To adequately address this issue, Gaul and Blanchette [5] introduced the so-called block ordering problem defined as follows. Given two partially assembled genomes, with each represented as an unordered set of blocks, the *block*

---

\* Supported in part by National Science Council of Republic of China under grant NSC100-2221-E-007-129-MY3.

*ordering problem* is to order and orient (i.e., assemble) the blocks of the two genomes such that the distance of genome rearrangements between the two assembled genomes is minimized. The blocks mentioned above are the so-called *contigs* (short for contiguous fragments), each of which can be represented by an ordered list of genes or markers. In their work [5], Gaul and Blanchette proposed a linear-time algorithm to solve the block ordering problem if the problem is further simplified to maximize the number of cycles in the breakpoint graph corresponding to the assembled genomes. The rationale behind this modification is totally based on a result obtained by Bourque and Pevzner [6], showing that the reversal distance between two assembled genomes can be approximated well by maximizing the number of cycles in their corresponding breakpoint graph. Actually, in addition to the number of cycles, the number of hurdles, as well as the presence of a fortress or not, is also important and needed for determining the actual reversal distance [7]. Therefore, it is still a challenge to efficiently solve the block ordering problem by optimizing the true rearrangement distance.

In the literature, many different kinds of genome rearrangements have been extensively studied [8], such as reversal (also called inversion), transposition, block-interchange (also called generalized transposition), translocation, fusion and fission. Reversal affects a segment on a chromosome by reversing this segment and exchanging its strands. Transposition rearranges a chromosome by interchanging its two adjacent and nonoverlapping segments. Block-interchange is a generalized transposition that exchanges two nonoverlapping but not necessarily adjacent segments on a chromosome. Translocation acts on two chromosomes by exchanging their end fragments. Both fusion and fission are special translocations, where fusion joins two chromosomes into one and fission splits a chromosome into two. In this study, we consider a variant of the block ordering problem, in which one of the two input genomes is still partially assembled but the other is completely assembled, with optimizing the block-interchange distance. For distinguishing this special block ordering problem from the original one, we call it as *one-sided block ordering problem*. In addition to genome rearrangements and phylogeny reconstruction, this one-sided block ordering problem has useful applications in genome resequencing [9], because the reference genome for a resequencing organism can serve as the completely assembled genome in the one-sided block problem so that the contigs of partially assembled genome for the resequencing organism can be assembled together.

In this study, we utilize permutation groups in algebra, instead of the breakpoint graphs used by Gaul and Blanchette [5], to design an efficient algorithm of  $\mathcal{O}(n \log n)$  time for solving the one-sided block ordering problem with block-interchange distance, where  $n$  is the number of genes or markers. In addition, we show that the assembly of the partially assembled genome can be done in  $\mathcal{O}(n)$  time and its block-interchange distance from the completely assembled genome can be calculated in advance in  $\mathcal{O}(n)$  time. In the rest of the paper, the terms “block” and “contig” are used interchangeably for convenience.

## 2 Preliminaries

In this study, we dedicate ourselves to linear, uni-chromosomal genomes. With a slight modification, however, our algorithmic result can still apply to circular, uni-chromosomal genomes, or to multi-chromosomal genomes with linear or circular chromosomes in a chromosome-by-chromosome manner. Once completely assembled, a linear, uni-chromosomal genome can be represented by a permutation of  $n$  integers between 1 and  $n$ , with each integer representing a gene or marker. If the genome is partially assembled, then it will be represented by an unordered set of blocks, where each block is an ordered list of some integers. Given an unordered set of blocks, say  $B$ , corresponding to a partially assembled genome, an *ordering* of  $B$  is an ordered list of its blocks, which *induces* a permutation of  $n$  integers by concatenating the blocks in this ordered list. In fact, the permutation induced by an ordering of  $B$  corresponds to an assembly of the blocks in  $B$ . For simplicity, we denote by  $\text{assembly}(B)$  the permutation induced by an optimal ordering of  $B$ . The one-sided block ordering problem we study in this paper is formally defined as follows:

### One-sided block ordering problem with block-interchange distance

**Input:** A partially assembled genome  $\pi$  and a completely assembled genome  $\sigma$ .

**Output:** Find an ordering of  $\pi$  such that the block-interchange distance between the permutation induced by this ordering of  $\pi$  and  $\sigma$  is minimized.

For our purpose, as shall be explained below, we use a cycle of some integers to represent a block of corresponding genes or markers and a product (respectively, addition) of cycles to represent an unordered (respectively, ordered) set of corresponding blocks. For example, let  $\pi = (1, 4)(3, 2)(5, 6)$  be a partially assembled genome with three unordered contigs and  $\sigma = (1, 2, \dots, 6)$  be a completely assembled genome with one contig. Then  $(1, 4, 5, 6, 3, 2)$  is the assembled genome induced by an ordering  $(1, 4) + (5, 6) + (3, 2)$  of  $\pi$ . It can be verified that this ordering is optimal because its induced genome  $(1, 4, 5, 6, 3, 2)$  can be transformed into  $\sigma$  by a block-interchange of blocks  $(4, 5, 6)$  and  $(2)$ .

Permutation groups have been proved to be a very useful tool in the studies of genome rearrangements [3,10,11,12,13]. Below, we recall some useful definitions, notations and properties borrowed from our previous work [12]. Basically, given a set  $E = \{1, 2, \dots, n\}$ , a *permutation* is defined to be a one-to-one function from  $E$  into itself and usually expressed as a product of cycles. For instance,  $\pi = (1)(3, 2)$  is a product of two cycles to represent a permutation of  $E = \{1, 2, 3\}$  and means that  $\pi(1) = 1$ ,  $\pi(2) = 3$  and  $\pi(3) = 2$ . The elements in a cycle can be arranged in any cyclic order and hence the cycle  $(3, 2)$  in the permutation exemplified above can be rewritten as  $(2, 3)$ . Moreover, if the cycles in a permutation are all disjoint (i.e., no common element in any two cycles), then the product of these cycles is called the *cycle decomposition* of the permutation. A cycle with  $k$  elements is further called a  $k$ -*cycle*. In convention, the 1-cycles in a permutation are not written explicitly since their elements are *fixed* in the permutation. If the cycles in a permutation are all 1-cycles, then this permutation is called an *identity permutation* and denoted by  $\mathbf{1}$ . Suppose that  $\alpha$  and  $\beta$  are two permutations of

$E$ . Then their product  $\alpha\beta$ , also called their *composition*, defines a permutation of  $E$  satisfying  $\alpha\beta(x) = \alpha(\beta(x))$  for all  $x \in E$ . If both  $\alpha$  and  $\beta$  are disjoint, then  $\alpha\beta = \beta\alpha$ . If  $\alpha\beta = \mathbf{1}$ , then  $\alpha$  is called the *inverse* of  $\beta$ , denoted by  $\beta^{-1}$ , and vice versa.

It is a fact that every permutation can be expressed into a product of 2-cycles (not necessarily disjoint), in which 1-cycles are still written implicitly. Given a permutation  $\alpha$  of  $E$ , its *norm*, denoted by  $\|\alpha\|$ , is defined to be the minimum number, say  $k$ , such that  $\alpha$  can be expressed as a product of  $k$  2-cycles. In the cycle decomposition of  $\alpha$ , let  $n_c(\alpha)$  denote the number of its disjoint cycles, notably including the 1-cycles not written explicitly. Given two permutations  $\alpha$  and  $\beta$  of  $E$ ,  $\alpha$  is said to *divide*  $\beta$ , denoted by  $\alpha|\beta$ , if and only if  $\|\beta\alpha^{-1}\| = \|\beta\| - \|\alpha\|$ . In our previous work [12], it has been shown that  $\|\alpha\| = |E| - n_c(\alpha)$  and for any  $k$  elements in  $E$ , say  $a_1, a_2, \dots, a_k$ , they all appear in a cycle of  $\alpha$  in the ordering of  $(a_1, a_2, \dots, a_k)|\alpha$ .

Let  $\alpha = (a_1, a_2)$  be a 2-cycle and  $\beta$  be an arbitrary permutation of  $E$ . If  $\alpha|\beta$ , that is, both  $a_1$  and  $a_2$  appear in a cycle of  $\beta$ , then the composition  $\alpha\beta$ , as well as  $\beta\alpha$ , has the effect of fission by breaking this cycle into two smaller cycles. For instance, let  $\alpha = (1, 3)$  and  $\beta = (1, 2, 3, 4)$ . Then  $\alpha|\beta$ , since both 1 and 3 are in the cycle  $(1, 2, 3, 4)$ , and  $\alpha\beta = (1, 2)(3, 4)$  and  $\beta\alpha = (4, 1)(2, 3)$ . On the other hand, if  $\alpha \nmid \beta$ , that is,  $a_1$  and  $a_2$  appear in different cycles of  $\beta$ , then  $\alpha\beta$ , as well as  $\beta\alpha$ , has the effect of fusion by joining the two cycles into a bigger cycle. For example, if  $\alpha = (1, 3)$  and  $\beta = (1, 2)(3, 4)$ , then  $\alpha \nmid \beta$  and, as a result,  $\alpha\beta = (1, 2, 3, 4)$  and  $\beta\alpha = (2, 1, 4, 3)$ . Intriguingly, a block-interchange acting on a cycle can be mimicked by two 2-cycles as described in the following lemma [12].

**Lemma 1 ([12]).** *Let  $\alpha$  denote a cycle and  $u, v, x$  and  $y$  be four elements in  $E$ . If  $(x, y)|\alpha$  and  $(u, v) \nmid (x, y)\alpha$ , then the effect of  $(u, v)(x, y)\alpha$  is a block-interchange acting on  $\alpha$ .*

### 3 Algorithmic Result

To clarify our algorithm, we start with defining some notations. For a contig  $\alpha$  with  $k$  genes, we denote it by  $\alpha = (\alpha[1], \alpha[2], \dots, \alpha[k])$  and, by convention, we call  $\alpha[1]$  as *tail* of  $\alpha$ . Let  $\pi = \pi_1\pi_2\dots\pi_m$  be a linear, uni-chromosomal genome that is partially assembled into  $m$  contigs  $\pi_1, \pi_2, \dots, \pi_m$ , and  $\sigma = (1, 2, \dots, n)$  be a linear, uni-chromosomal genome that is assembled completely. Assume that there are  $m_i$  genes in each contig  $\pi_i$ , where  $1 \leq i \leq m$ . Let  $C = \{c_k = n+k+1 : 0 \leq k \leq 2m-1\}$  be a set of  $2m$  caps, each of which is represented by an integer between  $n+1$  and  $n+2m$ . For the purpose of designing our algorithm later, we add two caps  $c_{2(i-1)}$  and  $c_{2(i-1)+1}$  to the ends of each contig  $\pi_i$ , leading to a capping contig  $\hat{\pi}_i$  with  $\hat{\pi}_i[1] = c_{2(i-1)}$  and  $\hat{\pi}_i[m_i + 2] = c_{2(i-1)+1}$ . Moreover, we insert  $m-1$  dummy contigs without any genes (i.e., null contigs)  $\sigma_2, \sigma_3, \dots, \sigma_m$  into  $\sigma$ , where  $\sigma$  becomes  $\sigma_1$ , and add two caps  $c_{2(i-1)}$  and  $c_{2(i-1)+1}$  to the ends of each contig  $\sigma_i$  to obtain a capping contig  $\hat{\sigma}_i$ , where  $\hat{\sigma}_i[1] = c_{2(i-1)}$  and  $\hat{\sigma}_i[m_i + 2] = c_{2(i-1)+1}$ . We denote the capping  $\pi$  and  $\sigma$

by  $\hat{\pi}$  and  $\hat{\sigma}$ , respectively. For example, assume that  $\pi = (1, 4)(3, 2)(5, 6)$  and  $\sigma = (1, 2, \dots, 6)$ . After capping, we have  $\hat{\pi} = (7, 1, 4, 8)(9, 3, 2, 10)(11, 5, 6, 12)$  and  $\hat{\sigma} = (7, 1, 2, \dots, 6, 8)(9, 10)(11, 12)$ . To distinguish the two caps in a capping contig, say  $\hat{\pi}_i$ , we call the left cap  $\hat{\pi}_i[1]$  as 5' cap and the right cap  $\hat{\pi}_i[m_i + 2]$  as 3' cap.

Given an integer  $x$  in  $E \cup C$ , we define a function  $\text{char}(x, \hat{\alpha})$  as follows to represent the character of  $x$  in a capping contig  $\hat{\alpha} = (\hat{\alpha}[1], \hat{\alpha}[2], \dots, \hat{\alpha}[k + 2])$ , which is obtained by adding two caps from  $C$  to the ends of a contig  $\alpha$  with  $k$  genes. Note that  $\hat{\alpha}$  contains  $x$ .

$$\text{char}(x, \hat{\alpha}) = \begin{cases} \text{C5, if } x = \hat{\alpha}[1], \\ \quad \text{that is, } x \text{ is the 5' cap in } \hat{\alpha}. \\ \text{C3, if } k \neq 0 \text{ and } x = \hat{\alpha}[k + 2], \\ \quad \text{that is, } \alpha \text{ is not null and } x \text{ is the 3' cap in } \hat{\alpha}. \\ \text{N3, if } k = 0 \text{ and } x = \hat{\alpha}[k + 2], \\ \quad \text{that is, } \alpha \text{ is null and } x \text{ is the 3' cap in } \hat{\alpha}. \\ \text{T, if } k \neq 0 \text{ and } x = \hat{\alpha}[2], \\ \quad \text{that is, } \alpha \text{ is not null and } x \text{ is the tail in } \alpha. \\ \text{O, otherwise.} \end{cases}$$

In addition, we define  $5\text{cap}(x, \hat{\alpha})$  to be the 5' cap of the capping contig  $\hat{\alpha}$ . For convenience, we extend the above definitions to a capping genome. For instance, given a capping genome, say  $\hat{\pi}$ ,  $\text{char}(x, \hat{\pi})$  denotes the character of  $x$  in a capping contig  $\hat{\pi}_i$  containing  $x$ , and  $5\text{cap}(x, \hat{\pi})$  denotes the 5' cap of  $\hat{\pi}_i$  (i.e.,  $\text{char}(x, \hat{\pi}) = \text{char}(x, \hat{\pi}_i)$  and  $5\text{cap}(x, \hat{\pi}) = 5\text{cap}(x, \hat{\pi}_i)$ ).

Basically, we deal with the contigs of a capping genome, say  $\hat{\pi}$ , as if they were linear chromosomes. Let  $c_1 = (x, y)$  and  $c_2 = (u, v)$  be two 2-cycles with character pairs of (non-C5, non-C5) and (C5, C5), respectively. In our previous study [12], we have shown that performing a translocation  $\tau$  on  $\hat{\pi}$  can be mimicked by the composition of  $c_2c_1\hat{\pi}$ , if  $(x, u)|\hat{\pi}$ ,  $(y, v)|\hat{\pi}$  and  $(x, y) \nmid \hat{\pi}$ , that is,  $x$  and  $u$ , as well as  $y$  and  $v$ , lie in the same contig in  $\hat{\pi}$ , but  $x$  and  $y$  occur in the different contigs in  $\hat{\pi}$ . Moreover, if the character pair of  $c_1$  is in  $\text{CEpair} = \{(C3, C3), (C3, N3), (T, T), (T, N3), (N3, N3)\}$ , then  $\tau$  acts on  $\hat{\pi}$  by exchanging a cap of some contig in  $\hat{\pi}$  with a cap of another contig and, as a result, leaves the original genome  $\pi$  unaffected. If  $c_1$  is a 2-cycle of character pair (T, C3) (respectively, (O, N3)), then  $\tau$  performing on  $\hat{\pi}$  becomes a fusion (respectively, fission) acting on  $\pi$ . It is not hard to see that the permutation induced by an ordering of the uncapped genome  $\pi$  can be considered as the result of applying consecutive  $m - 1$  fusions to the  $m$  contigs in  $\pi$ . Based on the above discussion, it can be realized that our purpose is to find  $m - 1$  translocations, each of which can be followed by one or more cap exchanges, to act on  $\hat{\pi}$  such that their rearrangement effects on the original  $\pi$  are  $m - 1$  fusions and the block-interchange distance between the resulting assembly of the contigs in  $\pi$  and  $\sigma$  is minimum.

Now, we describe our algorithm for efficiently solving the one-sided block ordering problem with block-interchange distance in the following Algorithm 1, where  $\delta$  denotes the block-interchange distance between  $\text{assembly}(\pi)$  and  $\sigma$ .

**Algorithm 1**

**1:** Let  $\sigma_1 = \sigma$  and add  $m - 1$  null contigs  $\sigma_2, \sigma_3, \dots, \sigma_m$  into  $\sigma$  such that  
 $\sigma = \sigma_1\sigma_2\dots\sigma_m$ .  
Obtain  $\hat{\pi} = \hat{\pi}_1\hat{\pi}_2\dots\hat{\pi}_m$  and  $\hat{\sigma} = \hat{\sigma}_1\hat{\sigma}_2\dots\hat{\sigma}_m$  by capping  $\pi$  and  $\sigma$ .

**2:** Compute  $\hat{\sigma}\hat{\pi}^{-1}$ .

**3: /\* To perform cap exchanges \*/**  
Let  $i = 0$ .  
**while** there are  $x$  and  $y$  in a cycle of  $\hat{\sigma}\hat{\pi}^{-1}$  such that  $(\text{char}(x, \hat{\pi}), \text{char}(y, \hat{\pi})) \in \text{CEpair}$  **do**  
  Let  $i = i + 1$ .  
  Find  $x$  and  $y$  in a cycle of  $\hat{\sigma}\hat{\pi}^{-1}$  with  $(\text{char}(x, \hat{\pi}), \text{char}(y, \hat{\pi})) \in \text{CEpair}$ .  
  Let  $\chi_i = (\text{5cap}(x, \hat{\pi}), \text{5cap}(y, \hat{\pi}))(x, y)$ .  
  Calculate new  $\hat{\pi} = \chi_i\hat{\pi}$  and new  $\hat{\sigma}\hat{\pi}^{-1} = \hat{\sigma}\hat{\pi}^{-1}\chi_i^{-1}$ .  
**end while**

**4: /\* To find consecutive  $m - 1$  fusions \*/**  
Let  $i = 0$ .  
**while** there are two adjacent integers  $x$  and  $y$  in a cycle of  $\hat{\sigma}\hat{\pi}^{-1}$  such that  
 $(\text{char}(x, \hat{\pi}), \text{char}(y, \hat{\pi})) = (\text{T}, \text{C3})$  and  $(x, y) \nmid \hat{\pi}$  **do**  
  Let  $i = i + 1$ .  
  Find two adjacent integers  $x$  and  $y$  in a cycle of  $\hat{\sigma}\hat{\pi}^{-1}$  such that  
 $(\text{char}(x, \hat{\pi}), \text{char}(y, \hat{\pi})) = (\text{T}, \text{C3})$  and  $(x, y) \nmid \hat{\pi}$ .  
  Let  $\tau_i = (\text{5cap}(x, \hat{\pi}), \text{5cap}(y, \hat{\pi}))(x, y)$ .  
  Calculate new  $\hat{\pi} = \tau_i\hat{\pi}$  and new  $\hat{\sigma}\hat{\pi}^{-1} = \hat{\sigma}\hat{\pi}^{-1}\tau_i^{-1}$ .  
**end while**  
**while**  $i < m - 1$  **do**  
  Let  $i = i + 1$ .  
  Find two adjacent integers  $x$  and  $y$  in a cycle of  $\hat{\sigma}\hat{\pi}^{-1}$  such that  
 $(\text{char}(x, \hat{\pi}), \text{char}(y, \hat{\pi})) = (\text{T}, \text{C3})$  and  $(x, y) \mid \hat{\pi}$ .  
  Find a contig in  $\hat{\pi}$  with at least a non-cap integer and its 3' cap, say  $z$ ,  
different from  $y$  (that is,  $\text{char}(z, \hat{\pi}) = \text{C3}$ ).  
  Let  $\tau_i = (y, z)(x, z)$ .  
  Calculate new  $\hat{\pi} = \tau_i\hat{\pi}$  and new  $\hat{\sigma}\hat{\pi}^{-1} = \hat{\sigma}\hat{\pi}^{-1}\tau_i^{-1}$ .  
**end while**  
Let  $\hat{\pi}_j$  be the capping contig with  $n + 2$  elements in current  $\hat{\pi}$ .  
Let  $\text{assembly}(\pi) = (\hat{\pi}_j[2], \hat{\pi}_j[3], \dots, \hat{\pi}_j[n + 1])$ .

**5: /\* To find block-interchanges \*/**  
Let  $\delta = 0$ .  
**while**  $\hat{\sigma}\hat{\pi}^{-1} \neq 1$  **do**  
  Let  $\delta = \delta + 1$ .  
  Choose any two adjacent integers  $x$  and  $y$  in a cycle of  $\hat{\sigma}\hat{\pi}^{-1}$ .  
  Find two adjacent integers  $u$  and  $v$  in a cycle of  $\hat{\sigma}\hat{\pi}^{-1}(x, y)$  such that  
 $(u, v) \nmid (x, y)\hat{\pi}$ .  
  Let  $\beta_\delta = (u, v)(x, y)$ .  
  Calculate new  $\hat{\pi} = \beta_\delta\hat{\pi}$  and new  $\hat{\sigma}\hat{\pi}^{-1} = \hat{\sigma}\hat{\pi}^{-1}\beta_\delta^{-1}$ .  
**end while**

**6:** Output  $\text{assembly}(\pi)$  and  $\delta$ .

We consider an example below to clarify the details of Algorithm 1. Let  $\pi = (1, 4)(3, 2)(5, 6)$  and  $\sigma = (1, 2, \dots, 6)$ . First of all, we add two null contigs into  $\sigma$  and cap all the contigs in  $\pi$  and  $\sigma$  in a way such that  $\hat{\pi} = (7, 1, 4, 8)(9, 3, 2, 10)(11, 5, 6, 12)$  and  $\hat{\sigma} = (7, 1, 2, \dots, 6, 8)(9, 10)(11, 12)$ . Next, we compute  $\hat{\sigma}\hat{\pi}^{-1} = (2, 4)(3, 10)(5, 12, 8)$ . It can be found that 12 and 8 are in a cycle of current  $\hat{\sigma}\hat{\pi}^{-1}$  with  $(\text{char}(12, \hat{\pi}), \text{char}(8, \hat{\pi})) = (\text{C3}, \text{C3})$ . We then perform a cap exchange on  $\hat{\pi}$  by multiplying  $(5\text{cap}(12, \hat{\pi}), 5\text{cap}(8, \hat{\pi}))(12, 8) = (11, 7)(12, 8)$  with  $\hat{\pi}$ , leading to new  $\hat{\pi} = (7, 1, 4, 12)(9, 3, 2, 10)(11, 5, 6, 8)$ . We also have new  $\hat{\sigma}\hat{\pi}^{-1} = (2, 4)(3, 10)(11, 7)(5, 12)$ . It can be observed that 5 and 12 are in the same cycle of  $\hat{\sigma}\hat{\pi}^{-1}$  with satisfying that  $\text{char}(5, \hat{\pi}) = \text{T}$ ,  $\text{char}(12, \hat{\pi}) = \text{C3}$  and  $(5, 12) \nmid \hat{\pi}$  (since 5 and 12 are in different contigs in current  $\hat{\pi}$ ). Hence, we perform a fusion on  $\hat{\pi}$ , by multiplying  $\tau_1 = (5\text{cap}(5, \hat{\pi}), 5\text{cap}(12, \hat{\pi}))(5, 12) = (11, 7)(5, 12)$  with  $\hat{\pi}$ , to obtain new  $\hat{\pi} = (7, 1, 4, 5, 6, 8)(9, 3, 2, 10)(11, 12)$ . We then have new  $\hat{\sigma}\hat{\pi}^{-1} = (2, 4)(3, 10)$ , in which 3 and 10 form a  $(\text{T}, \text{C3})$  pair but they belong to the same contig in  $\hat{\pi}$ , that is,  $(3, 10)|\hat{\pi}$ . In this case,  $\hat{\pi}$  has a contig  $(7, 1, 4, 5, 6, 8)$  whose 3' cap is 8 that is different from 10. Hence, we multiply  $(10, 8)(3, 8)$  with  $\hat{\pi}$  to obtain new  $\hat{\pi} = (7, 1, 4, 5, 6, 3, 2, 8)(9, 10)(11, 12)$  and new  $\hat{\sigma}\hat{\pi}^{-1} = (2, 4)(3, 8)$ . Finally, we can see that  $(2, 4)(3, 8)\hat{\pi} = \hat{\sigma}$ , meaning that we can transform  $\hat{\pi}$  into  $\hat{\sigma}$  using a block-interchange by multiplying  $(2, 4)(3, 8)$  with  $\hat{\pi}$ . As a result, we obtain an ordering  $(1, 4) + (5, 6) + (3, 2)$  of  $\pi$  whose induced permutation  $(1, 4, 5, 6, 3, 2)$  can be transformed into  $\sigma = (1, 2, \dots, 6)$  by a block-interchange, which exchanges the block  $(4, 5, 6)$  with the block  $(2)$  in  $(1, 4, 5, 6, 3, 2)$ .

Actually, the operation  $\tau_i = (y, z)(x, z)$  used in the step 4 of Algorithm 1 acts on  $\hat{\pi}$  still as a fusion of  $\pi$ , as explained as follows. Notice that  $(x, y)|\hat{\pi}$ , meaning that  $x$  and  $y$  are in the same cycle of  $\hat{\pi}$  and, therefore,  $5\text{cap}(x, \hat{\pi}) = 5\text{cap}(y, \hat{\pi})$ . As a result, we have  $(5\text{cap}(y, \hat{\pi}), 5\text{cap}(z, \hat{\pi}))(5\text{cap}(x, \hat{\pi}), 5\text{cap}(z, \hat{\pi})) = 1$ . This further implies that  $\tau_i$  can be rewritten as  $\tau_i = \alpha_2\alpha_1$ , where  $\alpha_1 = (5\text{cap}(x, \hat{\pi}), 5\text{cap}(z, \hat{\pi}))(x, z)$  and  $\alpha_2 = (5\text{cap}(y, \hat{\pi}), 5\text{cap}(z, \hat{\pi}))(y, z)$ . As will be clear later,  $\alpha_1$  acts on  $\hat{\pi}$  as a fusion of  $\pi$  and  $\alpha_2$  continues to act on  $\alpha_1\hat{\pi}$  as a cap exchange since  $\alpha_2 = (5\text{cap}(y, \alpha_1\hat{\pi}), 5\text{cap}(z, \alpha_1\hat{\pi}))(y, z)$ . As a result, the rearrangement effect of acting  $\tau_i$  on  $\hat{\pi}$  is still equivalent to a fusion acting on  $\pi$ . As discussed above, it can be realized that a fusion to  $\pi$  can be mimicked by a translocation  $\tau$ , which acts on  $\hat{\pi}$  as a fusion of  $\pi$ , followed by zero or more translocations, which act on  $\tau\hat{\pi}$  as cap exchanges.

In the following, we prove the correctness of Algorithm 1. Initially, it is not hard to see that all the 5' caps are fixed in  $\hat{\sigma}\hat{\pi}^{-1}$ . For any integer  $x$  with  $\text{char}(x, \hat{\pi}_i) = \text{T}$ , where  $1 \leq i \leq m$ , if  $\hat{\pi}^{-1}(x) \neq \hat{\sigma}_1[1]$ , that is, the 5' cap of  $\hat{\pi}_i$  is not equal to that of  $\hat{\sigma}_1$ , then the character of  $\hat{\sigma}\hat{\pi}^{-1}(x)$  in  $\hat{\pi}$  must be C3. If any cycle in  $\hat{\sigma}\hat{\pi}^{-1}$  contains any two elements  $x$  and  $y$  with the same character (either T or C3) in  $\hat{\pi}$ , then we can extract a 2-cycle  $(x, y)$  from  $\hat{\sigma}\hat{\pi}^{-1}$  and multiply  $(5\text{cap}(x, \hat{\pi}), 5\text{cap}(y, \hat{\pi}))(x, y)$  with  $\hat{\pi}$  to exchange the caps of the contigs containing  $x$  and  $y$ , respectively, in  $\hat{\pi}$ . This is the job to be performed in the step 3 in Algorithm 1. After finishing the cap exchanges in the step 3, each cycle in the remaining  $\hat{\sigma}\hat{\pi}^{-1}$  has at most one element with T character and at most one element with C3 character. In other words, after running the step 3,

there are exactly  $m - 1$  cycles in the resulting  $\hat{\sigma}\hat{\pi}^{-1}$  such that each such a cycle contains exactly one element, say  $x$ , with  $\text{char}(x, \hat{\pi}) = T$  and exactly one element, say  $y$ , with  $\text{char}(y, \hat{\pi}) = C3$ , and  $\hat{\sigma}\hat{\pi}^{-1}(x) = y$ . In this case, we can further derive  $(m - 1)$  2-cycles from these  $(m - 1)$  cycles in  $\hat{\sigma}\hat{\pi}^{-1}$  with each 2-cycle having a character pair of  $(T, C3)$ . Intriguingly, we shall show below that these  $(m - 1)$  2-cycles of  $(T, C3)$  pair, denoted by  $f_1, f_2, \dots, f_{m-1}$ , can be used to obtain an optimal ordering of  $\pi$  such that the block-interchange distance between its induced permutation  $\text{assembly}(\pi)$  and  $\sigma$  is minimum. The following lemma is straightforward according to our previous study [12].

**Lemma 2.** *Let  $c_1 = (x, y)$  be a 2-cycle with  $\text{char}(x, \hat{\pi}) = T$  and  $\text{char}(y, \hat{\pi}) = C3$ , and let  $c_2 = (5\text{cap}(x, \hat{\pi}), 5\text{cap}(y, \hat{\pi}))$ . If  $(x, y) \nmid \hat{\pi}$ , then the effect of  $c_2c_1\hat{\pi}$  is a fusion that acts on  $\pi$  by concatenating the contig containing  $y$  with the contig containing  $x$ .*

For  $1 \leq k \leq m - 1$ , we simply let  $f_k = (x_k, y_k)$ , where  $\text{char}(x_k, \hat{\pi}) = T$  and  $\text{char}(y_k, \hat{\pi}) = C3$ , and  $g_k = (5\text{cap}(x_k, \hat{\pi}), 5\text{cap}(y_k, \hat{\pi}))$ . As mentioned previously, the permutation induced by an ordering of  $\pi$  can be mimicked by performing  $m - 1$  consecutive fusions on  $\pi$  that has  $m$  contigs initially. According to Lemma 2, if  $f_k \nmid \hat{\pi}$ , where  $1 \leq k \leq m - 1$ , then  $f_k$ , as well as  $g_k$ , can be applied to  $\hat{\pi}$  to function as a fusion of two contigs in  $\pi$ . However, not all  $f_1, f_2, \dots, f_{m-1}$  have such a function. Suppose that only the first  $\lambda$  2-cycles  $f_1, f_2, \dots, f_\lambda$  can be used to serve as fusions acting on  $\pi$ , where  $0 \leq \lambda \leq m - 1$ , that is,  $f_k \nmid \hat{\pi}$  for  $1 \leq k \leq \lambda$ , but  $f_k \mid \hat{\pi}$  for  $\lambda + 1 \leq k \leq m - 1$ . In this situation, we shall show below that we still can use  $f_1, f_2, \dots, f_{m-1}$  to derive an optimal ordering of  $\pi$ , as we did in the step 4 in Algorithm 1.

Recall that the 5' caps are all fixed in the beginning  $\hat{\sigma}\hat{\pi}^{-1}$  (before the step 3 in our algorithm). As mentioned before, for any translocation used to perform on  $\hat{\pi}$ , it can be expressed as two 2-cycles, one with (non-C5, non-C5) character pair and the other with (C5, C5). It is not hard to verify that during the process of the step 3, no two elements  $x$  and  $y$  with  $\text{char}(x, \hat{\pi}) = C5$  but  $\text{char}(y, \hat{\pi}) \neq C5$  can be found in a cycle of the  $\hat{\sigma}\hat{\pi}^{-1}$ , that is, C5 and non-C5 elements are not mixed together in the same cycle of  $\hat{\sigma}\hat{\pi}^{-1}$ . Actually, this property still continues to be asserted when we later perform any translocation on  $\hat{\pi}$  to function as a fusion of  $\pi$ . Let us now pay attention on those cycles in  $\hat{\sigma}\hat{\pi}^{-1}$  with only non-C5 elements and temporarily denote the composition of these cycles by  $\phi(\hat{\sigma}\hat{\pi}^{-1})$ . If we still can find any two elements  $x$  and  $y$  from a cycle in  $\phi(\hat{\sigma}\hat{\pi}^{-1})$  such that  $(5\text{cap}(x), 5\text{cap}(y))(x, y)$  is an exchange of caps when applying it to  $\hat{\pi}$ , then we apply this cap exchange to  $\hat{\pi}$  until we cannot find any one from  $\phi(\hat{\sigma}\hat{\pi}^{-1})$ . Finally, we denote such a  $\phi(\hat{\sigma}\hat{\pi}^{-1})$  without any cap exchange by  $\psi(\hat{\sigma}\hat{\pi}^{-1})$ . Basically,  $\psi(\hat{\sigma}\hat{\pi}^{-1})$  can be considered as a permutation of  $E' = E \cup \{c_{2i-1} : 1 \leq i \leq m\}$  and hence its norm  $\|\psi(\hat{\sigma}\hat{\pi}^{-1})\|$  is equal to  $|E'| - n_c(\psi(\hat{\sigma}\hat{\pi}^{-1}))$ .

**Lemma 3.** *Let  $\tau = (5\text{cap}(x, \hat{\pi}), 5\text{cap}(y, \hat{\pi}))(x, y)$  with  $\text{char}(x, \hat{\pi}) = T$ ,  $\text{char}(y, \hat{\pi}) = C3$  and  $(x, y) \nmid \hat{\pi}$ . Then  $\|\psi(\hat{\sigma}\hat{\pi}^{-1})\| - \|\psi(\hat{\sigma}\hat{\pi}^{-1}\tau^{-1})\| \in \{-1, 0, 1\}$ .*

*Proof.* For simplicity, it is assumed that we cannot find any cap exchange from  $\hat{\sigma}\hat{\pi}^{-1}$  to perform on  $\hat{\pi}$ . We then consider the following two cases.

Case 1: Suppose that  $(x, y) \mid \hat{\sigma}\hat{\pi}^{-1}$ , that is, both  $x$  and  $y$  lie in the same cycle, say  $\alpha$ , in  $\hat{\sigma}\hat{\pi}^{-1}$ . Without loss of generality, let  $\alpha = (a_1, a_2, \dots, a_i \equiv x, \dots, a_j \equiv y)$ . Then  $\alpha$  can be expressed as  $\alpha = \alpha_1\alpha_2(x, y)$ , where  $\alpha_1 = (a_1, a_2, \dots, a_i)$  and  $\alpha_2 = (a_{i+1}, a_{i+2}, \dots, a_j)$ . Clearly, after applying  $\tau$  to  $\hat{\pi}$ , the cycle  $\alpha$  becomes two disjoint cycles  $\alpha_1$  and  $\alpha_2$  in  $\hat{\sigma}\hat{\pi}^{-1}\tau^{-1}$ . It means that  $n_c(\psi(\hat{\sigma}\hat{\pi}^{-1}\tau^{-1})) = n_c(\psi(\hat{\sigma}\hat{\pi}^{-1})) + 1$  and hence  $\|\psi(\hat{\sigma}\hat{\pi}^{-1})\| - \|\psi(\hat{\sigma}\hat{\pi}^{-1}\tau^{-1})\| = 1$ .

Case 2: Suppose that  $(x, y) \nmid \hat{\sigma}\hat{\pi}^{-1}$ , that is,  $x$  and  $y$  lie in the different cycles, say  $\alpha_1$  and  $\alpha_2$ , in  $\hat{\sigma}\hat{\pi}^{-1}$ . Then performing  $\tau$  on  $\hat{\pi}$  leads  $\alpha_1$  and  $\alpha_2$  to be joined together into a cycle, say  $\alpha$ , in  $\hat{\sigma}\hat{\pi}^{-1}\tau^{-1}$ . If  $\alpha_1$ , as well as  $\alpha_2$ , does not contain both T and C3 elements simultaneously, then  $n_c(\psi(\hat{\sigma}\hat{\pi}^{-1}\tau^{-1})) = n_c(\psi(\hat{\sigma}\hat{\pi}^{-1})) - 1$  and hence  $\|\psi(\hat{\sigma}\hat{\pi}^{-1})\| - \|\psi(\hat{\sigma}\hat{\pi}^{-1}\tau^{-1})\| = -1$ . If exactly one of  $\alpha_1$  and  $\alpha_2$  contains both T and C3 elements simultaneously, then joining  $\alpha_1$  and  $\alpha_2$  will also change  $\text{char}(x, \hat{\pi})$  from T to O and  $\text{char}(y, \hat{\pi})$  from C3 to N3. Therefore, the cycle  $\alpha$  contains a C3 (or T) element and an N3 element. In this case, we can use these two elements, along with their corresponding 5' caps in  $\hat{\pi}$ , as a cap exchange to perform on  $\hat{\pi}$ , resulting in that the cycle  $\alpha$  is divided into two smaller ones in new  $\hat{\sigma}\hat{\pi}^{-1}$ . As a result,  $n_c(\psi(\hat{\sigma}\hat{\pi}^{-1}\tau^{-1})) = n_c(\psi(\hat{\sigma}\hat{\pi}^{-1}))$  and hence  $\|\psi(\hat{\sigma}\hat{\pi}^{-1})\| - \|\psi(\hat{\sigma}\hat{\pi}^{-1}\tau^{-1})\| = 0$ . Suppose that both  $\alpha_1$  and  $\alpha_2$  contain T and C3 elements at the same time. Then, after applying  $\tau$  to  $\hat{\pi}$ , one of the above two T elements becomes an O element in new  $\hat{\pi}$ , leading to  $\alpha$  containing only a T element, as well as a C3 element and an N3 element. Next, we can use the T and N3 elements (or the C3 and N3 elements) in  $\alpha$  and their corresponding 5' caps in  $\hat{\pi}$  to exchange the caps of  $\hat{\pi}$ . After that,  $\alpha$  is divided into two cycles in the new  $\hat{\sigma}\hat{\pi}^{-1}$  and, consequently,  $n_c(\psi(\hat{\sigma}\hat{\pi}^{-1}\tau^{-1})) = n_c(\psi(\hat{\sigma}\hat{\pi}^{-1}))$  and hence  $\|\psi(\hat{\sigma}\hat{\pi}^{-1})\| - \|\psi(\hat{\sigma}\hat{\pi}^{-1}\tau^{-1})\| = 0$ .  $\square$

Note that if  $\hat{\pi} = \hat{\sigma}$ , then  $\|\psi(\hat{\sigma}\hat{\pi}^{-1})\| = 0$ . According to Lemmas 2 and 3, any translocation  $\tau$  that acts on  $\hat{\pi}$  as a fusion of  $\pi$  decreases the norm  $\|\psi(\hat{\sigma}\hat{\pi}^{-1})\|$  at most by one. Hence, we call  $\tau$  as a *good* fusion of  $\pi$  if  $\|\psi(\hat{\sigma}\hat{\pi}^{-1})\| - \|\psi(\hat{\sigma}\hat{\pi}^{-1}\tau^{-1})\| = 1$ . By the discussion in the proof of Lemma 3, we have the following corollary.

**Corollary 1.** *Let  $\tau = (5\text{cap}(x, \hat{\pi}), 5\text{cap}(y, \hat{\pi}))(x, y)$  with  $\text{char}(x, \hat{\pi}) = T$ ,  $\text{char}(y, \hat{\pi}) = C3$  and  $(x, y) \nmid \hat{\pi}$ . If  $(x, y) \mid \hat{\sigma}\hat{\pi}^{-1}$ , then  $\tau$  is a good fusion to perform on  $\pi$ .*

According to Corollary 1, it can be realized that each  $f_k$ , where  $1 \leq k \leq \lambda$ , along with  $g_k$  can serve as a good fusion to act on  $\pi$ . For simplicity, we assume that performing any  $g_i f_i$  on  $\hat{\pi}$  will not change the role of other  $g_j f_j$  as a good fusion of the resulting  $\pi$ , where  $1 \leq i, j \leq \lambda$ . If  $\lambda = m - 1$ , then performing the  $m - 1$  translocations  $g_1 f_1, g_2 f_2, \dots, g_{m-1} f_{m-1}$  on  $\hat{\pi}$ , as used in Algorithm 1, corresponds to an optimal ordering of  $\pi$  with an induced permutation  $\text{assembly}(\pi)$  such that the block-interchange distance between  $\text{assembly}(\pi)$  and  $\sigma$  is minimum. If  $\lambda < m - 1$ , then we show below that the fusions of  $m - 1$  contigs in  $\pi$  performed by our algorithm on the basis of  $f_1, f_2, \dots, f_{m-1}$  is still optimal. As mentioned before, a fusion to  $\pi$  can be mimicked by a translocation  $\tau$  acting on  $\hat{\pi}$  as a fusion of  $\pi$ , followed by zero or more translocations acting on  $\tau\hat{\pi}$  as cap exchanges. For convenience, therefore, we assume that any translocation  $\tau$  that acts on  $\hat{\pi}$  to serve as a fusion of  $\pi$  is followed zero or more cap exchanges to

further act on  $\tau\hat{\pi}$  such that no more cap exchange can be derived from  $\hat{\sigma}\hat{\pi}^{-1}$  in the following.

**Lemma 4.** *Let  $\tau_1, \tau_2, \dots, \tau_{m-1}$  be any sequence of  $m-1$  translocations that act on  $\hat{\pi}$  as fusions to assemble  $m-1$  contigs in  $\pi$ . Let  $\hat{\omega}_k$  be the genome obtained by performing  $\tau_k$  and zero or more following cap exchanges on  $\hat{\omega}_{k-1}$  such that no more cap exchange can be derived from  $\hat{\sigma}\hat{\omega}_k^{-1}$ , where  $\hat{\omega}_0 = \hat{\pi}$  and  $1 \leq k \leq m-1$ . Then  $\|\psi(\hat{\sigma}\hat{\omega}_0^{-1})\| - \|\psi(\hat{\sigma}\hat{\omega}_{m-1}^{-1})\| \leq \lambda$ .*

*Proof.* For simplicity, we assume that in the beginning, no cap exchange can be derived from  $\hat{\sigma}\hat{\omega}_0^{-1}$  to act on  $\hat{\omega}_0$ . Let  $\omega_k$  denote the genome obtained from  $\hat{\omega}_k$  by removing its caps. By Lemma 3,  $\|\psi(\hat{\sigma}\hat{\omega}_{k-1}^{-1})\| - \|\psi(\hat{\sigma}\hat{\omega}_k^{-1})\| \in \{-1, 0, 1\}$  and by Corollary 1,  $\|\psi(\hat{\sigma}\hat{\omega}_{k-1}^{-1})\| - \|\psi(\hat{\sigma}\hat{\omega}_k^{-1})\| = 1$  if  $\tau_k$  is a good fusion to  $\omega_{k-1}$ . In fact, there are at most  $\lambda$  translocations from  $\tau_1, \tau_2, \dots, \tau_{m-1}$  that are good fusions. The reason is as follows. Basically, as mentioned before, we can derive  $\lambda$  2-cycles  $f_1, f_2, \dots, f_\lambda$  from  $\hat{\sigma}\hat{\pi}^{-1}$  such that  $g_1f_1, g_2f_2, \dots, g_\lambda f_\lambda$  are good fusions to act on  $\pi$ , as well as other 2-cycles  $f_{\lambda+1}, f_{\lambda+2}, \dots, f_{m-1}$  that cannot derive any good fusions to act on  $\pi$  since their T and C3 elements are in the same contig in  $\hat{\pi}$ . If we can further extract a 2-cycle  $f = (x, y)$  from  $\hat{\sigma}\hat{\pi}^{-1}$  such that  $gf$  is a good fusion to act on  $\pi$ , where  $\text{char}(x, \hat{\pi}) = \text{T}$ ,  $\text{char}(y, \hat{\pi}) = \text{C3}$  and  $g = (\text{5cap}(x, \hat{\pi}), \text{5cap}(y, \hat{\pi}))$ , then the C3 element  $y$  must locate at a contig whose T element is in some  $f_k$ , where  $1 \leq k \leq \lambda$ . It implies that the good fusion  $gf$  cannot act on  $\pi$  together with  $g_1f_1, g_2f_2, \dots, g_\lambda f_\lambda$  at the same time, since the role of  $gf$  as a good fusion will be destroyed after performing  $g_1f_1, g_2f_2, \dots, g_\lambda f_\lambda$ . Suppose that  $\tau_1, \tau_2, \dots, \tau_{m-1}$  are the translocations obtained by the step 4 of Algorithm 1. That is,  $\tau_k = g_k f_k$  for  $1 \leq k \leq \lambda$  and  $\tau_k = (y_k, z_k)(x_k, z_k)$  for  $\lambda + 1 \leq k \leq m-1$ , where  $z_k$  is the 3' cap in any contig that is different from the contig containing  $y_k$ . Clearly, for  $1 \leq k \leq \lambda$ ,  $\|\psi(\hat{\sigma}\hat{\omega}_{k-1}^{-1})\| - \|\psi(\hat{\sigma}\hat{\omega}_k^{-1})\| = 1$  since  $\tau_k$  is a good fusion to  $\omega_{k-1}$ . Moreover, for  $\lambda + 1 \leq k \leq m-1$ ,  $\|\psi(\hat{\sigma}\hat{\omega}_{k-1}^{-1})\| - \|\psi(\hat{\sigma}\hat{\omega}_k^{-1})\| = 0$ , due to the following reason. First of all, we have  $\psi(\hat{\sigma}\hat{\omega}_k^{-1}) = \psi(\hat{\sigma}\hat{\omega}_{k-1}^{-1})\tau_k^{-1}$ , in which the composition of  $(x_k, y_k)\tau_k^{-1}$  equals to  $(x_k, y_k)(x_k, z_k)(y_k, z_k) = (x_k, z_k)$ , where notably  $(x_k, y_k)$  is extracted from a cycle in  $\psi(\hat{\sigma}\hat{\omega}_{k-1}^{-1})$ . It means that in  $\psi(\hat{\sigma}\hat{\omega}_k^{-1})$ ,  $y_k$  will be fixed, which increases the number of cycles by one, and the cycle  $(x_k, z_k)$  will join other two cycles respectively containing  $x_k$  and  $z_k$  together into one cycle, which decreases the number of cycles by one. As a result,  $n_c(\psi(\hat{\sigma}\hat{\omega}_k^{-1})) = n_c(\psi(\hat{\sigma}\hat{\omega}_{k-1}^{-1}))$  and  $\|\psi(\hat{\sigma}\hat{\omega}_{k-1}^{-1})\| - \|\psi(\hat{\sigma}\hat{\omega}_k^{-1})\| = 0$ . Therefore, we have  $\|\psi(\hat{\sigma}\hat{\omega}_0^{-1})\| - \|\psi(\hat{\sigma}\hat{\omega}_{m-1}^{-1})\| = \lambda$  for the  $(m-1)$  translocations obtained by the step 4 of Algorithm 1. In fact, to let  $\|\psi(\hat{\sigma}\hat{\omega}_0^{-1})\| - \|\psi(\hat{\sigma}\hat{\omega}_{m-1}^{-1})\| > \lambda$  happen, there must be a translocation  $\tau_i$  that acts on  $\hat{\omega}_{i-1}$  as a fusion of  $\omega_{i-1}$  satisfying either (1)  $\|\psi(\hat{\sigma}\hat{\omega}_{i-1}^{-1})\| - \|\psi(\hat{\sigma}\hat{\omega}_i^{-1})\| = 0$ , the number of good fusions newly created by  $\tau_i$  and its following cap exchanges minus that of good fusions currently destroyed by  $\tau_i$  and the following cap exchanges is greater than or equal to one, and the total available good fusions can assemble more contigs than before, or (2)  $\|\psi(\hat{\sigma}\hat{\omega}_{i-1}^{-1})\| - \|\psi(\hat{\sigma}\hat{\omega}_i^{-1})\| = -1$ , the number of good fusions created by  $\tau_i$  and its following cap exchanges minus that of the currently destroyed good fusions is greater than or equal to two, and the total good fusions can assemble

more contigs than before. However, we can show that no such a translocation  $\tau_i$  exists, where its details are omitted here due to space constraints. Therefore, we can conclude that  $\|\psi(\hat{\sigma}\hat{\omega}_0^{-1})\| - \|\psi(\hat{\sigma}\hat{\omega}_{m-1}^{-1})\| \leq \lambda$ .  $\square$

Based on Lemma 4, as well as the discussion in its proof, the  $m-1$  fusions derived by Algorithm 1 correspond to an optimal ordering of  $\pi$  with an induced permutation  $\text{assembly}(\pi)$  such that the block-interchange distance between  $\text{assembly}(\pi)$  and  $\sigma$  is minimized. This block-interchange distance is calculated in the step 5 in Algorithm 1, which is based on the algorithm in our previous study [3], and is equal to  $\frac{\|\hat{\sigma}\hat{\pi}^{-1}\|}{2}$ , where  $\hat{\pi}$  is the genome obtained by performing the cap exchanges and  $m-1$  fusions on the initial capping of  $\pi$ , as done in the steps 3 and 4 in Algorithm 1, respectively. The total time complexity of Algorithm 1 is  $\mathcal{O}(n \log n)$ . The reason is as follows. Since  $m \leq n$ , the cost of the step 1 for capping the input genomes is  $\mathcal{O}(n)$ . The computation of  $\hat{\sigma}\hat{\pi}^{-1}$  in the step 2 still can be done in  $\mathcal{O}(n)$  time. Recall that after running the step 3, each cycle in  $\hat{\sigma}\hat{\pi}^{-1}$  has at most a T element and at most a C3 element. Totally, there are  $m$  T elements and  $m$  C3 elements in the cycles of  $\hat{\sigma}\hat{\pi}^{-1}$ . Moreover, deriving a 2-cycle to serve as a cap exchange from a long cycle in  $\hat{\sigma}\hat{\pi}^{-1}$  will divide this long cycle into two smaller cycles. Hence, there are  $\mathcal{O}(n)$  cap exchanges to be performed in the step 3, which totally cost  $\mathcal{O}(n)$  time since each cap exchange needs only constant time. The step 4 assembles  $m$  contigs by utilizing  $(m-1)$  2-cycles  $f_1, f_2, \dots, f_{m-1}$ , which can be derived in advance from  $\hat{\sigma}\hat{\pi}^{-1}$  in  $\mathcal{O}(n)$  time. Since each fusion requires only constant time, the cost of the step 4 is  $\mathcal{O}(m+n)$ , which is equal to  $\mathcal{O}(n)$ . As to the step 5, it can be done in  $\mathcal{O}(\delta n)$  time, according to the analysis in our previous study [3]. In fact, this time complexity can be further improved into  $\mathcal{O}(n + \delta \log \delta)$  by using the algorithm we proposed in another previous work [11]. Since  $\delta \leq n$ , the cost of the step 5 is  $\mathcal{O}(n \log n)$ . As a result, the time complexity of Algorithm 1 is  $\mathcal{O}(n \log n)$ . Note that as mentioned above,  $\text{assembly}(\pi)$  can be obtained in  $\mathcal{O}(n)$  time after finishing the step 4 of Algorithm 1. In addition, the block-interchange distance  $\delta = \frac{\|\hat{\sigma}\hat{\pi}^{-1}\|}{2}$  between  $\text{assembly}(\pi)$  and  $\sigma$  can be computed in  $\mathcal{O}(n)$  time. It is worth mentioning that in spite of the input genomes  $\pi$  and  $\sigma$  being linear, uni-chromosomal genomes in our previous discussion, our Algorithm 1 still can apply to circular, uni-chromosomal genomes with a slight modification, or to the multi-chromosomal genomes with linear or circular chromosomes in chromosome-by-chromosome manner without increasing its time complexity.

**Theorem 1.** *Given a partially assembled genome  $\pi$  and a completely assembled genome  $\sigma$ , the one-sided block ordering problem with block-interchange distance can be solved in  $\mathcal{O}(n \log n)$  time. Moreover, an optimal ordering of  $\pi$  can be done in  $\mathcal{O}(n)$  time and the block-interchange distance between the permutation induced by the optimal ordering of  $\pi$  and  $\sigma$  is  $\frac{\|\hat{\sigma}\hat{\pi}^{-1}\|}{2}$  that can be computed in  $\mathcal{O}(n)$  time, where  $\hat{\pi}$  is the capping genome of  $\pi$  with the cap exchanges and  $m-1$  fusions being done,  $\hat{\sigma}$  is the capping genome of  $\sigma$  and  $n$  is the number of genes or markers.*

## 4 Conclusion

In this work, we introduced and studied the one-sided block problem with optimizing the block-interchange distance, which has useful applications in the studies of genome rearrangements, phylogeny reconstruction and genome resequencing. We finally designed an efficient algorithm to solve this problem in  $\mathcal{O}(n \log n)$  time, where  $n$  is the number of genes or markers. In addition, we showed that the assembly of the partially assembled genome can be done in  $\mathcal{O}(n)$  time and its block-interchange distance from the completely assembled genome can also be calculated in advance in  $\mathcal{O}(n)$  time. It would be an interesting future work to study and design efficient algorithms to solve the (one-sided) block problem with optimizing other rearrangement distances by considering other operations, such as reversals and/or translocations.

## References

- Shendure, J., Ji, H.L.: Next-generation DNA sequencing. *Nature Biotechnology* 26, 1135–1145 (2008)
- Metzker, M.L.: Sequencing technologies – the next generation. *Nature Reviews Genetics* 11, 31–46 (2010)
- Lin, Y.C., Lu, C.L., Chang, H.Y., Tang, C.Y.: An efficient algorithm for sorting by block-interchanges and its application to the evolution of vibrio species. *Journal of Computational Biology* 12, 102–112 (2005)
- Huang, Y.L., Huang, C.C., Tang, C.Y., Lu, C.L.: SoRT<sup>2</sup>: a tool for sorting genomes and reconstructing phylogenetic trees by reversals, generalized transpositions and translocations. *Nucleic Acids Research* 38, W221–W227 (2010)
- Gaul, É., Blanchette, M.: Ordering partially assembled genomes using gene arrangements. In: Bourque, G., El-Mabrouk, N. (eds.) RECOMB-CG 2006. LNCS (LNBI), vol. 4205, pp. 113–128. Springer, Heidelberg (2006)
- Bourque, G., Pevzner, P.A.: Genome-scale evolution: reconstructing gene orders in the ancestral species. *Genome Research* 12, 26–36 (2002)
- Hannenhalli, S., Pevzner, P.A.: Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *Journal of the ACM* 46, 1–27 (1999)
- Fertin, G., Labarre, A., Rusu, I., Tannier, E., Vialette, S.: Combinatorics of Genome Rearrangements. The MIT Press, Cambridge (2009)
- Bentley, D.R.: Whole-genome re-sequencing. *Current Opinion in Genetics and Development* 16, 545–552 (2006)
- Lu, C.L., Huang, Y.L., Wang, T.C., Chiu, H.T.: Analysis of circular genome rearrangement by fusions, fissions and block-interchanges. *BMC Bioinformatics* 7, 295 (2006)
- Huang, Y.L., Huang, C.C., Tang, C.Y., Lu, C.L.: An improved algorithm for sorting by block-interchanges based on permutation groups. *Information Processing Letters* 110, 345–350 (2010)
- Huang, Y.L., Lu, C.L.: Sorting by reversals, generalized transpositions, and translocations using permutation groups. *Journal of Computational Biology* 17, 685–705 (2010)
- Huang, K.H., Chen, K.T., Lu, C.L.: Sorting permutations by cut-circularize-linearize-and-paste operations. *BMC Genomics* 12, S26 (2011)

# A Combinatorial Approach for Multiple RNA Interaction: Formulations, Approximations, and Heuristics

Syed Ali Ahmed\*, Saad Mneimneh\*\*,\*\*\*,†, and Nancy L. Greenbaum‡

The Graduate Center and Hunter College, City University of New York (CUNY),  
New York, USA

sahmed3@gc.cuny.edu, {saad,ngreenba}@hunter.cuny.edu

**Abstract.** The interaction of two RNA molecules involves a complex interplay between folding and binding that warranted recent developments in RNA-RNA interaction algorithms. However, biological mechanisms in which more than two RNAs take part in an interaction exist.

We formulate multiple RNA interaction as a computational problem, which not surprisingly turns out to be NP-complete. Our experiments with approximation algorithms and heuristics for the problem suggest that this formulation is indeed useful to determine interaction patterns of multiple RNAs when information about which RNAs interact is not necessarily available (as opposed to the case of two RNAs where one must interact with the other), and because the resulting RNA structure often cannot be predicated by existing algorithms when RNAs are simply handled in pairs. We show instances of multiple RNA interaction that are accurately predicted by our algorithms.

**Keywords:** multiple RNA interaction, dynamic programming, approximation algorithms, structure prediction.

## 1 Introduction

The interaction of two RNA molecules has been independently formulated as a computational problem in several works, e.g. [1,2,3]. In their most general form, these formulations lead to NP-hard problems. To overcome this hurdle, researchers have been either reverting to approximation algorithms, or imposing algorithmic restrictions; for instance, analogous to the avoidance of pseudoknot formation in the folding of RNAs.

While these algorithms had limited use in the beginning, they became important venues for (and in fact popularized) an interesting biological fact: RNAs

---

\* Supported by NSF Award CCF-AF 1049902 and a CUNY GC Science Fellowship.

\*\* Supported by NSF Award CCF-AF 1049902 and in part by NIH Award GM07800.

\*\*\* Corresponding author.

† The first two authors contributed equally to this work.

‡ Supported by NSF Award MCB 0929394.

interact. For instance, micro-RNAs (miRNAs) bind to a complementary part of messenger RNAs (mRNAs) and inhibit their translation [4]. One might argue that such a simple interaction does not present a pressing need for RNA-RNA interaction algorithms; however, more complex forms of RNA-RNA interaction exist. In *E. Coli*, CopA binds to the ribosome binding site of CopT, also as a regulation mechanism to prevent translation [5]; so does OxyS to fhlA [6]. In both of these structures, the simultaneous folding (within the RNA) and binding (to the other RNA) are non-trivial to be predicted as separate events. To account for this, most of the RNA-RNA interaction algorithms calculate the probability for a pair of subsequences (one of each RNA) to participate in the interaction, and in doing so they generalize the energy model used for the partition function of a single RNA to the case of two RNAs [7,8,9,10,11,12]. This generalization takes into consideration the simultaneous aspect of folding and binding.

Not surprisingly, there exist other mechanisms in which more than two RNA molecules take part in an interaction. Typical scenarios involve the interaction of multiple small nucleolar RNAs (snoRNAs) with ribosomal RNAs (rRNAs) in guiding the methylation of the rRNAs [4], and multiple small nuclear RNAs (snRNA) with mRNAs in the splicing of introns [13]. Even with the existence of a computational framework for a single RNA-RNA interaction, it is reasonable to believe that interactions involving multiple RNAs are generally more complex to be treated pairwise. In addition, given a pool of RNAs, it is not trivial to predict which RNAs interact without some prior biological information.

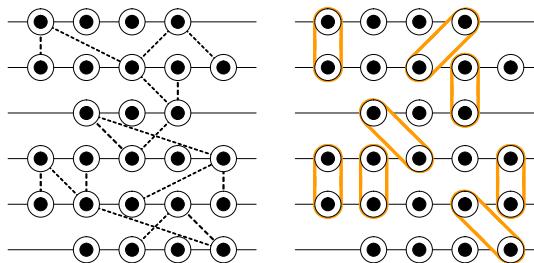
We formulate the problem of multiple RNA interaction by bringing forward an optimization perspective where each part of an RNA will contribute certain weights to the entire interaction when binding to different parts of other RNAs. We seek to maximize the total weight. This notion of weight can be obtained by using existing RNA-RNA interaction algorithms on pairs of RNAs. We call our formulation the Pegs and Rubber Bands problem. We show that under certain restrictions, which are similar to those against pseudoknots, the problem remains NP-hard (in fact it becomes equivalent to a special instance of the interaction of two RNAs). We describe a polynomial time approximation scheme PTAS for the problem, some heuristics, and experimental results. For instance, given a pool of RNAs in which interactions between pairs of RNAs are known, our algorithm is capable of identifying those pairs and predicting satisfactorily the pattern of interaction between them [8]. Moreover, our algorithm finds the correct interaction of a given instance of splicing consisting of two snRNAs (a modified U2-U6 human snRNA complex) and two structurally autonomous parts of an intron [14], a total of four RNAs. When (partially) mixing the two examples in one pool, our algorithm structurally separates them.

## 2 Pegs and Rubber Bands: A Formulation

We introduce an optimization problem we call Pegs and Rubber Bands that will serve a base framework for the multiple RNA interaction problem. The link

between the two problems will be made shortly after the description of Pegs and Rubber Bands.

Consider  $m$  levels numbered 1 to  $m$  with  $n_l$  pegs in level  $l$  numbers 1 to  $n_l$ . There is an infinite supply of rubber bands that can be placed around two pegs in consecutive levels. For instance, we can choose to place a rubber band around peg  $i$  in level  $l$  and peg  $j$  in level  $l+1$ ; we call it a rubber band at  $[l, i, j]$ . Every such pair of pegs  $[l, i]$  and  $[l+1, j]$  contribute their own weight  $w(l, i, j)$ . The Pegs and Rubber Bands problem is to maximize the total weight by placing rubber bands around pegs in such a way that no two rubber bands intersect. In other words, each peg can have at most one rubber band around it, and if a rubber band is placed at  $[l, i_1, j_1]$  and another at  $[l, i_2, j_2]$ , then  $i_1 < i_2 \Leftrightarrow j_1 < j_2$ . We assume without loss of generality that  $w(l, i, j) \neq 0$  to avoid the unnecessary placement of rubber bands and, therefore, either  $w(l, i, j) > 0$  or  $w(l, i, j) = -\infty$ . Figure 1 shows an example.



**Fig. 1.** Pegs and Rubber Bands. All positive weights are equal to 1 and are represented by dashed lines. The optimal solution achieves a total weight of 8.

Given an optimal solution, it can always be reconstructed from left to right by repeatedly placing some rubber band at  $[l, i, j]$  such that, at the time of this placement, no rubber band is around peg  $[l, k]$  for  $k > i$  and no rubber band is around peg  $[l+1, k]$  for  $k > j$ . This process can be carried out by a dynamic programming algorithm to compute the maximum weight (in exponential time), by defining  $W(i_1, i_2, \dots, i_m)$  to be the maximum weight when we truncate the levels at pegs  $[1, i_1], [2, i_2], \dots, [m, i_m]$  (see Figure 2). The maximum weight is given by  $W(n_1, n_2, \dots, n_m)$  and the optimal solution can be obtained by standard backtracking. When all levels have  $O(n)$  pegs, this algorithm runs in  $O(mn^m)$  time and  $O(n^m)$  space.

## 2.1 Multiple RNA Interaction as Pegs and Rubber Bands

To provide some initial context we now describe how the formulation of Pegs and Rubber Bands, though in a primitive way, captures the problem of multiple RNA interaction. We think of each level as an RNA and each peg as one base of the RNA. The weight  $w(l, i, j)$  corresponds to the negative of the energy

$$W(i_1, i_2, \dots, i_m) = \max \left\{ \begin{array}{l} W(i_1 - 1, i_2, \dots, i_m) \\ W(i_1, i_2 - 1, i_3, \dots, i_m) \\ \vdots \\ W(i_1, \dots, i_{m-1}, i_m - 1) \\ W(i_1 - 1, i_2 - 1, i_3, \dots, i_m) + w(1, i_1, i_2) \\ W(i_1, i_2 - 1, i_3 - 1, i_4, \dots, i_m) + w(2, i_2, i_3) \\ \vdots \\ W(i_1, \dots, i_{m-2}, i_{m-1} - 1, i_m - 1) + w(m - 1, i_{m-1}, i_m) \end{array} \right\}$$

where  $W(0, 0, \dots, 0) = 0$ .

**Fig. 2.** Dynamic programming algorithm for Pegs and Rubber Bands

contributed by the binding of the  $i^{\text{th}}$  base of RNA  $l$  to the  $j^{\text{th}}$  base of RNA  $l+1$ . It should be clear, therefore, that an optimal solution for Pegs and Rubber Bands represents the lowest energy conformation in a base-pair energy model, when a pseudoknot-like restriction is imposed on the RNA interaction (rubber bands cannot intersect). In doing so, we obviously assume that an order on the RNAs is given with alternating sense and antisense, and that the first RNA interacts with the second RNA, which in turn interacts with the third RNA, and so on. We later relax this ordering and condition on the interaction pattern of the RNAs. While a simple base-pairing model is not likely to give realistic results, our goal for the moment is simply to establish a correspondence between the two problems.

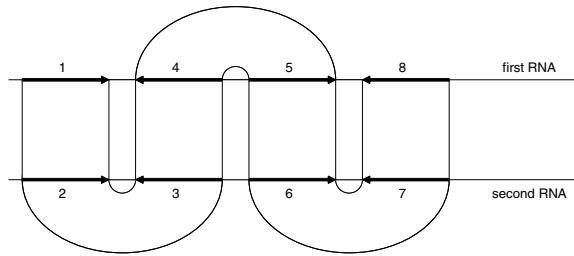
## 2.2 Complexity of the Problem and Approximations

With the above correspondence in mind, the problem of Pegs and Rubber Bands can be viewed as a instance of a classical RNA-RNA interaction, involving only two RNAs that is: We construct the first as RNA 1 followed by RNA 4 reversed followed by RNA 5 followed by RNA 8 reversed and so on, and the second as RNA 2 followed by RNA 3 reversed followed by RNA 6 followed by RNA 7 reversed and so on, as shown in Figure 3.

Therefore, Pegs and Rubber Bands can be solved as an RNA-RNA interaction problem. While this RNA-RNA interaction represents a restricted instance of the more general NP-hard problem, it is still NP-hard. In fact, Pegs and Rubber Bands itself is NP-hard.

**Theorem 1.** *Pegs and Rubber Bands is NP-hard.*

*Proof:* We make a reduction from the longest common subsequence (LCS) for a set of binary strings, which is an NP-hard problem. In this reduction, pegs are labeled and  $w(l, i, j)$  depends only on the label of peg  $[l, i]$  and the label of peg  $[l+1, j]$ . We describe this weight as a function of labels shortly. Each binary string is modified by adding the symbol  $b$  between every two consecutive bits. A string of original length  $n$  is then transformed into two consecutive (identical) levels of  $2n - 1$  pegs each, where each peg is labeled by the corresponding symbol in



**Fig. 3.** Pegs and Rubber Bands as a special instance of RNA-RNA interaction, vertical lines indicate regions where only interaction (binding of the two RNAs) is allowed, and curved lines indicate regions where only folding within each RNA is allowed

$\{0, 1, b\}$ . For any given integer  $k$ , the first and last levels consist of  $k$  pegs labeled  $*$ . We now define the weight as a function of labels:  $w(0, 0) = w(1, 1) = w(b, b) = w(*, 0) = w(*, 1) = w(0, *) = w(1, *) = 1$  and  $w(x, y) = -\infty$  otherwise. It is easy to verify that the strings have a common subsequence of length  $k$  if and only if the optimal solution has a weight of  $\sum_i (2n_i - 1) + k = 2 \sum_i n_i - m + k$  (when every peg has a rubber band around it), where  $n_i$  is the original length of string  $i$  and  $m$  is the number of strings. ■

```

* * * *
| | | |
0b0b1b0b1b1b1
|| | | | || ||
0b0b1b0b1b1b1
| | | |
0b1b0b1b0
| | | ||
0b1b0b1b0
| | | |
1b0b0b1b0b1
|||| | | |
1b0b0b1b0b1
| | | |
* * * *

```

**Fig. 4.** Reduction from LCS for  $\{0010111, 01010, 100101\}$  to Pegs and Rubber Bands (the symbol  $|$  denotes a rubber band). The optimal solution with weight  $2(7 + 5 + 6) - 3 + 4 = 37$  corresponds to a common subsequence of length 4, namely 0101.

While our problem is NP-hard, we can show that the same formulation can be adapted to obtain a polynomial time approximation. A maximization problem admits a polynomial time approximation scheme (PTAS) iff for every fixed  $\epsilon > 0$  there is an algorithm with a running time polynomial in the size of the input

that finds a solution within  $(1 - \epsilon)$  of optimal [15]. We show below that we can find a solution within  $(1 - \epsilon)$  of optimal in time  $O(m\lceil\frac{1}{\epsilon}\rceil n^{\lceil\frac{1}{\epsilon}\rceil})$ , where  $m$  is the number of levels and each level has  $O(n)$  pegs.

**Theorem 2.** *Pegs and Rubber Bands admits a PTAS.*

*Proof:* Let  $OPT$  be the weight of the optimal solution and denote by  $W[i \dots j]$  the weight of the optimal solution when the problem is restricted to levels  $i, i+1, \dots, j$  (a sub-problem). For a given  $\epsilon > 0$ , let  $k = \lceil\frac{1}{\epsilon}\rceil$ . Consider the following  $k$  solutions (weights), each obtained by a concatenation of optimal solutions for sub-problems consisting of at most  $k$  levels.

$$W_1 = W[1 \dots 1] + W[2 \dots k+1] + W[k+2 \dots 2k+1] + \dots$$

$$W_2 = W[1 \dots 2] + W[3 \dots k+2] + W[k+3 \dots 2k+2] + \dots$$

⋮

$$W_k = W[1 \dots k] + W[k+1 \dots 2k] + W[2k+1 \dots 3k] + \dots$$

While each  $W_i \leq OPT$ , it is easy to verify that every pair of consecutive levels appear in exactly  $k-1$  of the above sub-problems. Therefore,

$$\begin{aligned} \sum_{i=1}^k W_i &\geq (k-1)OPT \\ \Rightarrow \max_i W_i &\geq \frac{k-1}{k} OPT \geq (1-\epsilon)OPT \end{aligned}$$

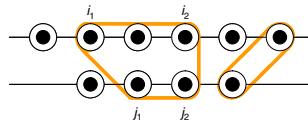
If  $m$  is the total number of levels, then there are  $O(m)$  sub-problems of at most  $k$  levels each and, therefore, the running time required to find  $\max_i W_i$  when every level has  $O(n)$  pegs is  $O(mkn^k) = O(m\lceil\frac{1}{\epsilon}\rceil n^{\lceil\frac{1}{\epsilon}\rceil})$ . ■

For a given integer  $k$ , the  $(1 - 1/k)$ -factor approximation algorithm is to simply choose the best  $W_i = W[1 \dots i] + W[i+1 \dots i+k] + W[i+k+1 \dots i+2k] + \dots$  as a solution, where  $W[i \dots j]$  denotes the weight of the optimal solution for the sub-problem consisting of levels  $i, i+1, \dots, j$ . However, as a practical step, and instead of using the  $W_i$ 's for the comparison, we can fill in for each  $W_i$  some additional rubber bands (interactions) between (RNAs) level  $i$  and level  $i+1$ , between level  $i+k$  and level  $i+k+1$ , and so on, by identifying the pegs of these levels (regions of RNAs) that are not part of the solution. This does not affect the theoretical guarantee but gives a larger weight to the solution. We call it *gap filling*.

### 3 Windows and Gaps: A Better Formulation for RNA Interaction

In the previous section, we described our initial attempt to view the interaction of  $m$  RNAs as a Pegs and Rubber Bands problem with  $m$  levels, where the first

RNA interacts with the second RNA, and the second with the third, and so on (so they alternate in sense and antisense). This used a simple base-pair energy model, which is not too realistic. We now address this issue (and leave the issues of the ordering and the interaction pattern to Section 3.3). A better model for RNA interaction will consider windows of interaction instead of single bases. In terms of our Pegs and Rubber Bands problem, this translates to placing rubber bands around a stretch of contiguous pegs in two consecutive levels, e.g. around pegs  $[l, i_1]$ ,  $[l, i_2]$ ,  $[l+1, j_1]$ , and  $[l+1, j_2]$ , where  $i_2 \geq i_1$  and  $j_2 \geq j_1$ . The weight contribution of placing such a rubber band is now given by  $w(l, i_2, j_2, u, v)$ , where  $i_2$  and  $j_2$  are the last two pegs covered by the rubber band in level  $l$  and level  $l+1$ , and  $u = i_2 - i_1 + 1$  and  $v = j_2 - j_1 + 1$  represent the length of the two windows covered in level  $l$  and level  $l+1$ , respectively.



**Fig. 5.** A rubber band can now be placed around a window of pegs, here  $u = 3$  and  $v = 2$  in the big window.

As a *heuristic*, we also allow for the possibility of imposing a gap  $g \geq 0$  between windows to establish a distance at which windows may be considered energetically separate. This gap is also taken into consideration when we perform the gap filling procedure described at the end of Section 3.1. The modified algorithm is shown in Figure 6, and if we set  $u = v = 1$  and  $g = 0$ , then we retrieve the original algorithm of Figure 2.

$$W(i_1, i_2, \dots, i_m) = \max \left\{ \begin{array}{l} W(i_1 - 1, i_2, \dots, i_m) \\ W(i_1, i_2 - 1, i_3, \dots, i_m) \\ \vdots \\ W(i_1, \dots, i_{m-1}, i_m - 1) \\ W((i_1 - u - g)^+, (i_2 - v - g)^+, i_3, \dots, i_m) + w(1, i_1, i_2, u, v) \\ W(i_1, (i_2 - u - g)^+, (i_3 - v - g)^+, i_4, \dots, i_m) + w(2, i_2, i_3, u, v) \\ \vdots \\ W(i_1, \dots, i_{m-2}, (i_{m-1} - u - g)^+, (i_m - v - g)^+) + \\ w(m-1, i_{m-1}, i_m, u, v) \end{array} \right\}$$

where  $x^+$  denotes  $\max(0, x)$ ,  $w(l, i, j, u, v) = -\infty$  if  $u > i$  or  $v > j$ ,  $0 < u, v \leq w$  (the maximum window size),  $g \geq 0$  (the gap), and  $W(0, 0, \dots, 0) = 0$ .

**Fig. 6.** Modified dynamic programming algorithm for Pegs and Rubber Bands with the windows and gaps formulation

The running time of the modified algorithm is  $O(mw^2n^m)$  and  $O(mw^2\lceil\frac{1}{\epsilon}\rceil n^{\lceil\frac{1}{\epsilon}\rceil})$  for the approximation scheme, where  $w$  is the maximum window length. If we impose that  $u = v$ , then those running times become  $O(mwn^m)$  and  $O(mw\lceil\frac{1}{\epsilon}\rceil n^{\lceil\frac{1}{\epsilon}\rceil})$  respectively.

For the correctness of the algorithm, we now have to assume that windows are *sub-additive*. In other words, we require the following condition (otherwise, the algorithm may compute an incorrect optimum due to the possibility of achieving the same window by two or more smaller ones with higher total weight):

$$\begin{aligned} w(l, i, j, u_1, v_1) + w(l, i - u_1, j - v_1, u_2, v_2) \\ \leq w(l, i, j, u_1 + u_2, v_1 + v_2) \end{aligned}$$

In our experience, most existing RNA-RNA interaction algorithms produce weights (the negative of the energy values) of RNA interaction windows that mostly conform to the above condition. In rare cases, we filter the windows to eliminate those that are not sub-additive. For instance, if the above condition is not met, we set  $w(l, i, j, u_1, v_1) = w(l, i - u_1, j - v_1, u_2, v_2) = -\infty$  (recursively starting with smaller windows).

## 4 Interaction Pattern and Permutations: A Heuristic

We now describe how to relax the ordering and the condition on the interaction pattern of the RNAs. We first identify each RNA as being *even* (sense) or *odd* (antisense), but this convention can obviously be switched. Given  $m$  RNAs and a permutation on the set  $\{1, \dots, m\}$ , we map the RNAs onto the levels of a Pegs and Rubber Bands problem as follows: We place the RNAs in the order in which they appear in the permutation on the same level as long as they have the same parity (they are either all even or all odd). We then increase the number of levels by one, and repeat. RNAs that end up on the same level are *virtually* considered as one RNA that is the concatenation of all. However, in the corresponding Pegs and Rubber Bands problem, we do not allow windows to span multiple RNAs, nor do we enforce a gap between two windows in different RNAs. For example, if we consider the following permutation of RNAs  $\{1, 3, 4, 7, 5, 8, 2, 6\}$ , where the RNA number also indicates its parity (for the sake of illustration), then we end up with the following placement: RNA 1 and RNA 3 in that order on the first level, followed by RNA 4 on the second level, followed by RNA 7 and RNA 5 in that order on the third level, followed by RNA 8, RNA 2, and RNA 6 in that order on the fourth level, resulting in four virtual RNAs on four levels of pegs as shown in Figure 7.

But what is the best placement as a Pegs and Rubber Bands problem for a given set of RNAs? Figure 8 shows a possible (greedy) heuristic that tackles this question by starting with a random permutation and then searching for the best one via neighboring permutations (and recall that the permutation uniquely determines the placement).

```

---RNA 1---RNA 3---
---RNA 4---
---RNA 7---RNA 5---
---RNA 8---RNA 2---RNA 6---

```

**Fig. 7.** Placement of the permutation  $\{1, 3, 4, 7, 5, 8, 2, 6\}$  where the RNA number also indicates its parity. The interaction pattern is less restrictive than before; for instance, RNA 7 can interact with RNA 2, RNA 4, RNA 6, and RNA 8.

Given  $\epsilon = 1/k$  and  $m$  RNAs  
 produce a random permutation  $\pi$  on  $\{1, \dots, m\}$   
 let  $W$  be the weight of the  $(1 - \epsilon)$ -optimal solution given  $\pi$   
**repeat**  
 better $\leftarrow$ false  
 generate a set  $\Pi$  of neighboring permutations for  $\pi$   
**for** every  $\pi' \in \Pi$  (in any order)  
 let  $W'$  be the weight of the  $(1 - \epsilon)$ -optimal solution given  $\pi'$   
**if**  $W' > W$   
**then**  $W \leftarrow W'$   
 $\pi \leftarrow \pi'$   
 better $\leftarrow$ true  
**until** **not** better

**Fig. 8.** A heuristic for multiple RNA interaction using the PTAS algorithm

To generate neighboring permutations for this heuristic algorithm one could adapt a standard 2-opt method used in the Traveling Salesman Problem (or other techniques). For instance, given permutation  $\pi$ , a neighboring permutation  $\pi'$  can be obtained by dividing  $\pi$  into three parts and making  $\pi'$  the concatenation of the first part, the reverse of the second part, and the third part. In other words, if  $\pi = (\alpha, \beta, \gamma)$ , then  $\pi' = (\alpha, \beta^R, \gamma)$  is a neighbor of  $\pi$ , where  $\beta^R$  is the reverse of  $\beta$ .

## 5 Experimental Results

We apply the algorithm of Section 3.3 using the 2-opt method, where the PTAS is based on the Windows and Gaps formulation of Section 3.2, with windows satisfying  $2 \leq u, v \leq w = 26$  (RNAUp's default [7]) and a gap  $g = 4$ . The weights  $w(l, i, j, u, v)$  are obtained from RNAUp as (negative of energy values):

$$\begin{aligned}
 w(l, i, j, u, v) \propto & \log p_l(i - u + 1, i) + \log p_{l+1}(j - v + 1, j) \\
 & + \log Z_l^I(i - u + 1, i, j - v + 1, j)
 \end{aligned}$$

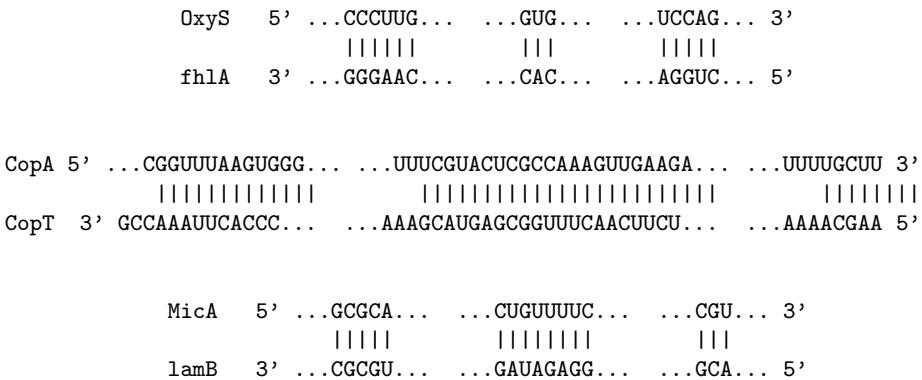
where  $p_l(i_1, i_2)$  is the probability that subsequence  $[i_1, i_2]$  is free (does not fold) in RNA  $l$ , and  $Z_l^I(i_1, i_2, j_1, j_2)$  is the partition function of the interaction of

subsequences  $[i_1, i_2]$  in RNA  $l$  and  $[j_1, j_2]$  in RNA  $l + 1$  (subject to no folding within RNAs).

The windows are filtered for sub-additivity as described in Section 3.2. We impose the condition that  $u = v$  for every window. We also have the option to compress RNAs on level  $l$  by removal of a base  $i$  whenever  $w(l, i, j, u, u)$  is less than some threshold for every  $j$  and every  $u$ ; however, peg  $[l, i]$  can still be part of some window, e.g. if  $w(l, i + x, j, x + y, x + y)$  is added to the solution, where  $x, y > 0$ . We did not use that option here. We pick the largest weight solution among several runs of the algorithm. The value of  $k$  and the gap filling criterion depend on the scenario, as described below.

## 5.1 Fishing for Pairs

Six RNAs of which three pairs are known to interact are used [8]. We are interested in identifying the three pairs. For this purpose, it will suffice to set  $k = 2$  and to ignore gap filling. Furthermore, we only consider solutions in which each RNA interacts with at most one other RNA. The solution with the largest weight identifies the three pairs correctly (Figure 9). In addition, the interacting sites in each pair are consistent (not surprisingly) with the predictions of existing RNA-RNA interaction algorithms, e.g. [10].

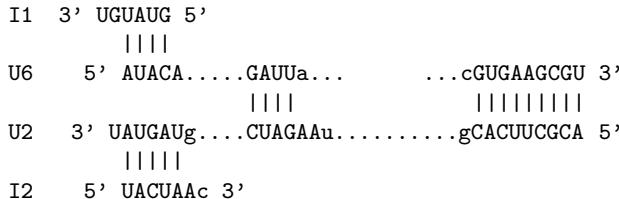


**Fig. 9.** Known pairs of interacting RNAs with reasonable solutions

## 5.2 Structure Prediction

The human snRNA complex U2-U6 is necessary for the splicing of a specific mRNA intron [14]. Only the preserved regions of the intron are considered, which consist of two structurally autonomous parts, resulting in an instance with a total of four RNAs. The algorithm is performed with  $k = 2, 3, 4$  and gap filling. In all three cases, the solution with the largest weight consistently finds the structure shown in Figure 10. This structure reveals a correct pattern

described in [13,14], and cannot be easily predicted by considering the RNAs in pairs; for instance, AUAC in U6 will bind to UAUG in both U2 and I1, and it is not immediately obvious which one to break without a global view, e.g. that AUGAU in U2 binds with UACUA in I2 as well. This is a typical issue of using local information to produce a globally optimal solution.



**Fig. 10.** A modified human snRNA U2-U6 complex in the splicing of an intron, as reported in [14]. Bases indicated by small letters are missing from the interaction. From left to right: g-c and a-u are missing due to the condition  $2 \leq u = v \leq 26$ , but also due to the added instability of a bulge loop when this condition is relaxed; c-g ends up being not favored by RNAup. I1 is shifted (UGU should interact with ACA instead) but this is a computational artifact of optimization that is hard to avoid. Overall, the structure is accurate and cannot be predicted by a pairwise handling of the RNAs.

### 5.3 Structural Separation

Six RNAs are used: CopA, CopT, and the four RNAs of the previous scenario. The algorithm is performed with  $k = 3$  and gap filling. The solution with the largest weight results in a successful prediction that separates the RNA complex CopA-CopT of Figure 9 from the RNA structure shown in Figure 10.

### 5.4 Making Improvements

In this section, we try to eliminate some heuristics, an approach we did not attempt in a previous work on the subject [16]. We relax the condition that  $u = v$  so we allow arbitrary window sizes and, furthermore, we drop the gap heuristic so we set  $g = 0$ . Some unwanted interactions now start to appear.

To correct for this, we modify RNAup weights in a reasonable way. For simplicity of notation, let  $A$  denote the subsequence  $[i - u + 1, i]$  in RNA  $l$  and  $B$  the subsequence  $[j - v + 1, j]$  in RNA  $l + 1$ . We now have

$$w(l, i, j, u, v) \propto \log p_A + \log p_B + \log q_A + \log q_B - \log(1 - p_{AB}^I)$$

where  $p_A$  and  $p_B$  are as before the probabilities that the corresponding subsequences are free,  $Z^I$  is replaced by  $(1 - p_{AB}^I)^{-1}$ , and  $p_{AB}^I$  is the probability that the two subsequences will interact (as opposed to individually fold) given

they are free (in the following,  $Z_A$  is the partition function for folding subsequence  $A$ ).

$$p_{AB}^I = \frac{Z_{AB}^I}{Z_{AB}^I + Z_A Z_B}$$

The probabilities  $q_A$  and  $q_B$  are additional corrective factors that reflect the preferential choice of the subsequences given they will interact.

$$q_A = \frac{p_B p_{AB}^I}{\sum_X p_X p_{AX}^I} \quad q_B = \frac{p_A p_{AB}^I}{\sum_Y p_Y p_{YB}^I}$$

where  $X$  and  $Y$  are subsequences in RNA  $l$  and RNA  $l+1$  respectively.

With these newly defined weights, we obtain similar results for Section 5.1 and the exact same results for Section 5.2.

## 6 Conclusion

While RNA-RNA interaction algorithms exist, they are not suitable for predicting RNA structures in which more than two RNA molecules interact. For instance, the interaction pattern may not be known, in contrast to the case of two RNAs where one must interact with the other. Moreover, even with some existing knowledge on the pattern of interaction, treating the RNAs pairwise may not lead to the best global structure. In this work, we formulate multiple RNA interaction as an optimization problem, prove it is NP-complete, and provide approximation and heuristic algorithms. We explore three scenarios: 1) fishing for pairs: given a pool of RNAs, we identify the pairs that are known to interact; 2) structure prediction: we predict a correct complex of two snRNAs (modified human U2 and U6) and two structurally autonomous parts of an intron, a total of four RNAs; and 3) structural separation: we successfully divide the RNAs into independent groups of multiple interacting RNAs.

## References

1. Pervouchine, D.D.: Iris: Intermolecular RNA interaction search. In: 15th International Conference on Genome Informatics (2004)
2. Alkan, C., Karakoc, E., Nadeau, J.H., Sahinalp, S.C., Zhang, K.: RNA-RNA interaction prediction and antisense RNA target search. Journal of Computational Biology 13(2) (2006)
3. Mneimneh, S.: On the approximation of optimal structures for RNA-RNA interaction. IEEE/ACM Transactions on Computational Biology and Bioinformatics (2009)
4. Meyer, I.M.: Predicting novel RNA-RNA interactions. Current Opinions in Structural Biology 18 (2008)
5. Kolb, F.A., Malmgren, C., Westhof, E., Ehresmann, C., Ehresmann, B., Wagner, E.G.H., Romby, P.: An unusual structure formed by antisense-target RNA binding involves an extended kissing complex with a four-way junction and a side-by-side helical alignment. RNA Society (2000)

6. Argaman, L., Altuvia, S.: fhla repression by oxys: Kissing complex formation at two sites results in a stable antisense-target RNA complex. *Journal of Molecular Biology* 300 (2000)
7. Muckstein, U., Tafer, H., Hackermuller, J., Bernhart, S.H., Stadler, P.F., Hofacker, I.L.: Thermodynamics of RNA-RNA binding. *Journal of Bioinformatics* (2006)
8. Chitsaz, H., Backofen, R., Sahinalp, S.C.: biRNA: Fast RNA-RNA binding sites prediction. In: Salzberg, S.L., Warnow, T. (eds.) WABI 2009. LNCS, vol. 5724, pp. 25–36. Springer, Heidelberg (2009)
9. Chitsaz, H., Salari, R., Sahinalp, S.C., Backofen, R.: A partition function algorithm for interacting nucleic acid strands. *Journal of Bioinformatics* (2009)
10. Salari, R., Backofen, R., Sahinalp, S.C.: Fast prediction of RNA-RNA interaction. *Algorithms for Molecular Biology* 5(5) (2010)
11. Huang, F.W.D., Qin, J., Reidys, C.M., Stadler, P.F.: Partition function and base pairing probabilities for RNA-RNA interaction prediction. *Journal of Bioinformatics* 25(20) (2009)
12. Li, A.X., Marz, M., Qin, J., Reidys, C.M.: RNA-RNA interaction prediction based on multiple sequence alignments. *Journal of Bioinformatics* (2010)
13. Sun, J.S., Manley, J.L.: A novel U2-U6 snRNA structure is necessary for mammalian mRNA splicing. *Genes and Development* 9 (1995)
14. Zhao, C., Bachu, R., Popovic, M., Devany, M., Brenowitz, M., Schlatterer, J.C., Greenbaum, N.L.: Conformational heterogeneity of the protein-free human spliceosomal U2-U6 snRNA complex. *RNA* 19, 561–573 (2013), doi:10.1261/rna.038265.113; These two autors contributed equally to the manuscript
15. Cormen, T., Leiserson, C.E., Rivest, R.L., Stein, C.: Approximation Algorithms in Introduction to Algorithms. MIT Press (2010)
16. Mneimneh, S., Ahmed, S.A., Greenbaum, N.L.: Multiple RNA interaction: Formulations, approximations, and heuristics. In: Fourth International Conference on Bioinformatics Models, Methods, and Algorithms (2013)

# Maximum Balanced Subgraph Problem Parameterized above Lower Bound

Robert Crowston, Gregory Gutin, Mark Jones, and Gabriele Muciaccia

Royal Holloway, University of London, UK

{robert,gutin,markj,G.Muciaccia}@cs.rhul.ac.uk

**Abstract.** We consider graphs without loops or parallel edges in which every edge is assigned + or -. Such a signed graph is balanced if its vertex set can be partitioned into parts  $V_1$  and  $V_2$  such that all edges between vertices in the same part have sign + and all edges between vertices of different parts have sign - (one of the parts may be empty). It is well-known that every connected signed graph with  $n$  vertices and  $m$  edges has a balanced subgraph with at least  $\frac{m}{2} + \frac{n-1}{4}$  edges and this bound is tight. We consider the following parameterized problem: given a connected signed graph  $G$  with  $n$  vertices and  $m$  edges, decide whether  $G$  has a balanced subgraph with at least  $\frac{m}{2} + \frac{n-1}{4} + \frac{k}{4}$  edges, where  $k$  is the parameter.

We obtain an algorithm for the problem of runtime  $8^k (kn)^{O(1)}$ . We also prove that for each instance  $(G, k)$  of the problem, in polynomial time, we can either solve  $(G, k)$  or produce an equivalent instance  $(G', k')$  such that  $k' \leq k$  and  $|V(G')| = O(k^3)$ . Our first result generalizes a result of Crowston, Jones and Mnich (ICALP 2012) on the corresponding parameterization of Max Cut (when every edge of  $G$  has sign -). Our second result generalizes and significantly improves the corresponding result of Crowston, Jones and Mnich for MaxCut: they showed that  $|V(G')| = O(k^5)$ .

## 1 Introduction

We consider undirected graphs with no parallel edges or loops and in which every edge is labelled by + or -. We call such graphs *signed graphs*, and edges, labelled by + and -, *positive* and *negative* edges, respectively. The labels + and - are the *signs* of the corresponding edges. Signed graphs are well-studied due to their various applications and interesting theoretical properties, see, e.g., [1,6,9,10,11,12,17].

Let  $G = (V, E)$  be a signed graph and let  $V = V_1 \cup V_2$  be a partition of the vertex set of  $G$  (i.e.,  $V_1 \cap V_2 = \emptyset$ ). We say that  $G$  is  $(V_1, V_2)$ -balanced if an edge with both endpoints in  $V_1$ , or both endpoints in  $V_2$  is positive, and an edge with one endpoint in  $V_1$  and one endpoint in  $V_2$  is negative;  $G$  is *balanced* if it is  $(V_1, V_2)$ -balanced for some partition  $V_1, V_2$  of  $V$  ( $V_1$  or  $V_2$  may be empty).

In some applications, we are interested in finding a maximum-size balanced subgraph of a signed graph [1,6,12,17]. We will call this problem SIGNED MAX CUT. This problem is a generalization of MAX CUT and as such is NP-hard (SIGNED MAX CUT is equivalent to MAX CUT when all edges of  $G$  are negative). Hüffner *et al.* [12] parameterized SIGNED MAX CUT below a tight upper bound: decide whether  $G = (V, E)$

contains a balanced subgraph with at least  $|E| - k$  edges, where  $k$  is the parameter<sup>1</sup>. Hüffner *et al.* [12] showed that this parameterized problem is fixed-parameter tractable (FPT) using a simple reduction to the EDGE BIPARTITION PROBLEM: decide whether an unsigned graph can be made bipartite by deleting at most  $k$  edges ( $k$  is the parameter). Using this result and a number of heuristic reductions, Hüffner *et al.* [12] designed a nontrivial practical algorithm that allowed them to exactly solve several instances of SIGNED MAX CUT that were previously solved only approximately by DasGupta *et al.* [6].

In this paper, we consider a different parameterization of SIGNED MAX CUT: decide whether a connected signed graph  $G$  with  $n$  vertices and  $m$  edges contains a subgraph with at least  $\frac{m}{2} + \frac{n-1}{4} + \frac{k}{4}$  edges<sup>2</sup>, where  $k$  is the parameter. Note that  $\text{pt}(G) = \frac{m}{2} + \frac{n-1}{4}$  is a tight lower bound on the number of edges in a balanced subgraph of  $G$  (this fact was first proved by Poljak and Turzík [15], for a different proof, see [2]). Thus, we will call this parameterized problem SIGNED MAX CUT ABOVE TIGHT LOWER BOUND or SIGNED MAX CUT ATLB. Whilst the parameterization of Hüffner *et al.* of MAX CUT ATLB is of interest when the maximum number of edges in a balanced subgraph  $H$  of  $G$  is close to the number of edges of  $G$ , SIGNED MAX CUT ATLB is of interest when the maximum number of edges in  $H$  is close to the minimum possible value in a signed graph on  $n$  vertices and  $m$  edges. Thus, the two parameterizations treat the opposite parts of the SIGNED MAX CUT “spectrum.”

It appears that it is much harder to prove that SIGNED MAX CUT ATLB is FPT than to show that the parameterization of Hüffner *et al.* of SIGNED MAX CUT is. Indeed, SIGNED MAX CUT ATLB is a generalization of the same parameterization of MAX CUT (denoted by MAX CUT ATLB) and the parameterized complexity of the latter was an open problem for many years (and was stated as an open problem in several papers) until Crowston *et al.* [5] developed a new approach for dealing with such parameterized problems<sup>3</sup>. This approach was applied by Crowston *et al.* [3] to solve an open problem of Raman and Saurabh [16] on maximum-size acyclic subgraph of an oriented graph. Independently, this problem was also solved by Mnich *et al.* [13] who obtained the solution as a consequence of a meta-theorem which shows that several graph problems parameterized above a lower bound of Poljak and Turzík [15] are FPT under certain conditions.

While the meta-theorem is for both unlabeled and labeled graphs, all consequences of the meta-theorem in [13] are proved only for parameterized problems restricted to unlabelled graphs. A possible reason is that one of the conditions of the meta-theorem requires us to show that the problem under consideration is FPT on a special family of graphs, called almost forests of cliques<sup>4</sup>. The meta-theorem is useful when it is relatively easy to find an FPT algorithm on almost forests of cliques. However, for SIGNED

<sup>1</sup> We use standard terminology on parameterized algorithmics, see, e.g., [7,8,14].

<sup>2</sup> We use  $\frac{k}{4}$  instead of just  $k$  to ensure that  $k$  is integral.

<sup>3</sup> Recall that MAX CUT is a special case of SIGNED MAX CUT when all edges are negative.

<sup>4</sup> Forests of cliques are defined in the next section. An almost forest of cliques is obtained from a forest of cliques by adding to it a small graph together with some edges linking the small graph with the forest of cliques.

MAX CUT ATLB it is not immediately clear what an FPT algorithm would be even on a clique.

Our attempts to check that SIGNED MAX CUT ATLB is FPT on almost forests of cliques led us to reduction rules that are applicable not only to almost forests of cliques, but to arbitrary instances of SIGNED MAX CUT ATLB. Thus, we found two alternatives to prove that SIGNED MAX CUT ATLB is FPT: with and without the meta-theorem. Since the first alternative required stating the meta-theorem and all related notions and led us to a slightly slower algorithm than the second alternative, we decided to use the second alternative.

We reduce an arbitrary instance of SIGNED MAX CUT ATLB to an instance which is an almost forest of cliques, but with an important additional property which allows us to make use of a slight modification of a dynamic programming algorithm of Crowston *et al.* [5] for MAX CUT ATLB on almost forests of cliques.

Apart from showing that MAX CUT ATLB is FPT, Crowston *et al.* [5] proved that the problem admits a kernel with  $O(k^5)$  vertices. They also found a kernel with  $O(k^3)$  vertices for a variation of MAX CUT ATLB, where the lower bound used is weaker than the Poljak-Turzík bound. They conjectured that a kernel with  $O(k^3)$  vertices exists for MAX CUT ATLB as well. In the main result of this paper, we show that SIGNED MAX CUT ATLB, which is a more general problem, also admits a polynomial-size kernel and, moreover, our kernel has  $O(k^3)$  vertices. Despite considering a more general problem than in [5], we found a proof which is shorter and simpler than the one in [5]; in particular, we do not use the probabilistic method. An  $O(k^3)$ -vertex kernel for SIGNED MAX CUT ATLB does not immediately imply an  $O(k^3)$ -vertex kernel for MAX CUT ATLB, but the same argument as for SIGNED MAX CUT ATLB shows that MAX CUT ATLB admits an  $O(k^3)$ -vertex kernel. This confirms the conjecture above.

## 2 Terminology, Notation and Preliminaries

For a positive integer  $l$ ,  $[l] = \{1, \dots, l\}$ . A cycle  $C$  in  $G$  is called *positive* (*negative*) if the number of negative edges in  $C$  is even (odd)<sup>5</sup>. The following characterization of balanced graphs is well-known.

**Theorem 1.** [11] *A signed graph  $G$  is balanced if and only if every cycle in  $G$  is positive.*

Let  $G = (V, E)$  be a signed graph. For a subset  $W$  of  $V$ , the  $W$ -switch of  $G$  is the signed graph  $G_W$  obtained from  $G$  by changing the signs of the edges between  $W$  and  $V \setminus W$ . Note that a signed graph  $G$  is balanced if and only if there exists a subset  $W$  of  $V$  ( $W$  may coincide with  $V$ ) such that  $G_W$  has no negative edges. Indeed, if  $G_W$  has no negative edges,  $G$  is  $(W, V \setminus W)$ -balanced. If  $G$  is  $(V_1, V_2)$ -balanced, then  $G_{V_1}$  has no negative edges.

Deciding whether a signed graph is balanced is polynomial-time solvable.

**Theorem 2.** [9] *Let  $G = (V, E)$  be a signed graph. Deciding whether  $G$  is balanced is polynomial-time solvable. Moreover, if  $G$  is balanced then, in polynomial time, we can find a subset  $W$  of  $V$  such that  $G_W$  has no negative edges.*

---

<sup>5</sup> To obtain the sign of  $C$  simply compute the product of the signs of its edges.

For a signed graph  $G$ ,  $\beta(G)$  will denote the maximum number of edges in a balanced subgraph of  $G$ . Furthermore, for a signed graph  $G = (V, E)$ ,  $\text{pt}(G)$  denotes the Poljak-Turzík bound:  $\beta(G) \geq \text{pt}(G)$ . If  $G$  is connected, then  $\text{pt}(G) = \frac{|E(G)|}{2} + \frac{|V(G)|-1}{4}$ , and if  $G$  has  $t$  components, then  $\text{pt}(G) = \frac{|E(G)|}{2} + \frac{|V(G)|-t}{4}$ . It is possible to find, in polynomial time, a balanced subgraph of  $G$  of size at least  $\text{pt}(G)$  [15].

The following easy property will be very useful in later proofs. It follows from Theorem 1 by observing that for a signed graph the Poljak-Turzík bound does not depend on the signs of the edges and that, for any cycle in  $G$ , the sign of the cycle in  $G$  and in  $G_W$  is the same.

**Corollary 1.** *Let  $G = (V, E)$  be a signed graph and let  $W \subset V$ . Then  $\text{pt}(G_W) = \text{pt}(G)$  and  $\beta(G_W) = \beta(G)$ .*

For a vertex set  $X$  in a graph  $G$ ,  $G[X]$  denotes the subgraph of  $G$  induced by  $X$ . For disjoint vertex sets  $X, Y$  of graph  $G$ ,  $E(X, Y)$  denotes the set of edges between  $X$  and  $Y$ . A *bridge* in a graph is an edge that, if deleted, increases the number of connected components of the graph. A *block* of a graph is either a maximal 2-connected subgraph or a connected component containing only one vertex.

For an edge set  $F$  of a signed graph  $G$ ,  $F^+$  and  $F^-$  denote the set of positive and negative edges of  $F$ , respectively. For a signed graph  $G = (V, E)$ , the *dual* of  $G$  is the signed graph  $\bar{G} = (\bar{V}, \bar{E})$ , where  $\bar{E}^+ = E^-$  and  $\bar{E}^- = E^+$ . A cycle in  $G$  is *dually positive* (*dually negative*) if the same cycle in  $\bar{G}$  is positive (negative).

For a graph  $G = (V, E)$ , the *neighborhood*  $N_G(W)$  of  $W \subseteq V$  is defined as  $\{v \in V : vw \in E, w \in W\} \setminus W$ ; the vertices in  $N_G(W)$  are called *neighbors* of  $W$ . If  $G$  is a signed graph, the *positive* neighbors of  $W \subseteq V$  are the neighbors of  $W$  in  $G^+ = (V, E^+)$ ; the set of positive neighbors is denoted  $N_G^+(W)$ . Similarly, for the *negative* neighbors and  $N_G^-(W)$ .

The next theorem is the ‘dual’ of Theorem 1, in the sense that it is its equivalent formulation on the dual of a graph.

**Theorem 3.** *Let  $G = (V, E)$  be a signed graph. Then the dual graph  $\bar{G}$  is balanced if and only if  $G$  does not contain a dually negative cycle.*

In the next sections, the notion of *forest of cliques* introduced in [5] plays a key role. A connected graph is a *tree of cliques* if the vertices of every cycle induce a clique. A *forest of cliques* is a graph whose components are trees of cliques. It follows from the definition that in a forest of cliques any block is a clique.

Note that a forest of cliques is a *chordal graph*, i.e., a graph in which every cycle has a chord, that is an edge between two vertices which are not adjacent in the cycle. The next lemma is a characterization of chordal graphs which have a balanced dual. A *triangle* is a cycle with three edges.

**Corollary 2.** *[<sup>6</sup>] Let  $G = (V, E)$  be a signed chordal graph. Then  $\bar{G}$  is balanced if and only if  $G$  does not contain a positive triangle.*

---

<sup>6</sup> Proofs of results marked by [ $\star$ ] can be found in [4].

**Corollary 3.** Let  $(G = (V, E), k)$  be an instance  $\mathcal{I}$  of SIGNED MAX CUT ATLB, let  $X \subseteq V(G)$  and let  $G[X]$  be a chordal graph which does not contain a positive triangle. Then there exists a set  $W \subseteq X$ , such that  $\tilde{\mathcal{I}} = (G_W, k)$  is equivalent to  $\mathcal{I}$ , and  $G_W[X]$  does not contain positive edges.

*Proof.* By Corollary 2,  $\bar{G}[X]$  is balanced: hence, by definition of balanced graph, there exists  $W \subseteq X$  such that  $\bar{G}_W[X]$  contains only positive edges, which means that  $G_W[X]$  contains only negative edges. By Corollary 1,  $(G_W, k)$  is an instance equivalent to the original one.  $\square$

Lastly, the next lemmas describe useful properties of MAX CUT ATLB which still hold for SIGNED MAX CUT ATLB.

**Lemma 1.** Let  $G = (V, E)$  be a connected signed graph and let  $V = U \cup W$  such that  $U \cap W = \emptyset$ ,  $U \neq \emptyset$  and  $W \neq \emptyset$ . Then  $\beta(G) \geq \beta(G[U]) + \beta(G[W]) + \frac{1}{2}|E(U, W)|$ . In addition, if  $G[U]$  has  $c_1$  components,  $G[W]$  has  $c_2$  components,  $\beta(G[U]) \geq \text{pt}(G[U]) + \frac{k_1}{4}$  and  $\beta(G[W]) \geq \text{pt}(G[W]) + \frac{k_2}{4}$ , then  $\beta(G) \geq \text{pt}(G) + \frac{k_1+k_2-(c_1+c_2-1)}{4}$ .

*Proof.* Let  $H$  ( $F$ ) be a balanced subgraph of  $G[U]$  ( $G[W]$ ) with maximum number of edges and let  $H$  ( $F$ ) be  $(U_1, U_2)$ -balanced ( $(W_1, W_2)$ -balanced). Let  $E_1 = E^+(U_1, W_1) \cup E^+(U_2, W_2) \cup E^-(U_1, W_2) \cup E^-(U_2, W_1)$  and  $E_2 = E(U, W) \setminus E_1$ . Observe that both  $E(H) \cup E(F) \cup E_1$  and  $E(H) \cup E(F) \cup E_2$  induce balanced subgraphs of  $G$  and the largest of them has at least  $\beta(G[U]) + \beta(G[W]) + \frac{1}{2}|E(U, W)|$  edges.

Now, observe that  $\text{pt}(G) = \text{pt}(G[U]) + \text{pt}(G[W]) + \frac{1}{2}|E(U, W)| + \frac{c_1+c_2-1}{4}$ . Hence  $\beta(G) \geq \text{pt}(G) + \frac{k_1+k_2-(c_1+c_2-1)}{4}$ .  $\square$

**Lemma 2.** [★] Let  $G = (V, E)$  be a signed graph,  $v \in V$  a cutvertex,  $Y$  a connected component of  $G - v$  and  $G' = G - Y$ . Then  $\text{pt}(G) = \text{pt}(G[V(Y) \cup \{v\}]) + \text{pt}(G')$  and  $\beta(G) = \beta(G[V(Y) \cup \{v\}]) + \beta(G')$ .

### 3 Fixed-Parameter Tractability

In this section, we prove that SIGNED MAX CUT ATLB is FPT by designing an algorithm of running time<sup>7</sup>  $O^*(8^k)$ . This algorithm is a generalization of the FPT algorithm obtained in [5] to solve MAX CUT ATLB. Given an instance  $(G = (V, E), k)$  of MAX CUT ATLB, the algorithm presented in [5] applies some reduction rules that either answer YES for MAX CUT ATLB or produce a set  $S$  of at most  $3k$  vertices such that  $G - S$  is a forest of cliques.

A key idea of this section is that it is possible to extend these rules such that we include into  $S$  at least one vertex for every dually negative cycle of  $G$ . As a result, Theorem 3 ensures that solving SIGNED MAX CUT ATLB on  $G - S$  is equivalent to solving MAX CUT ATLB. Therefore, it is possible to guess a partial solution on  $S$  and then solve MAX-CUT-WITH-WEIGHTED-VERTICES<sup>8</sup> on  $G - S$ . Since a forest of

<sup>7</sup> In the  $O^*$ -notation widely used in parameterized algorithmics, we omit not only constants, but also polynomial factors.

<sup>8</sup> This problem is defined just before Theorem 5.

cliques is a chordal graph, Corollary 2 implies that it is enough to put into  $S$  at least one vertex for every positive triangle in  $G$  (instead of every dually negative cycle). Our reduction rules are inspired by the rules used in [5], but our rules are more involved in order to deal with positive triangles.

The rules apply to an instance  $(G, k)$  of SIGNED MAX CUT ATLB and output an instance  $(G', k')$  where  $G'$  is obtained by deleting some vertices of  $G$ . In addition, the rules can mark some of the deleted vertices: marked vertices will form the set  $S$  such that  $G - S$  is a forest of cliques. Note that every time a rule marks some vertices, it also decreases the parameter  $k$ .

The instance  $(G', k')$  that the rules produce does not have to be equivalent to  $(G, k)$ , but it has the property that if it is a YES-instance, then  $(G, k)$  is a YES-instance too. For this reason, these rules are called *one-way* reduction rules [3].

Note that in the description of the rules,  $G$  is a connected signed graph, and  $C$  and  $Y$  denote connected components of a signed graph such that  $C$  is a clique which does not contain a positive triangle.

**Reduction Rule 1.** *If  $abca$  is a positive triangle such that  $G - \{a, b, c\}$  is connected, then mark  $a, b, c$ , delete them and set  $k' = k - 3$ .*

**Reduction Rule 2.** *If  $abca$  is a positive triangle such that  $G - \{a, b, c\}$  has two connected components  $C$  and  $Y$ , then mark  $a, b, c$ , delete them, delete  $C$ , and set  $k' = k - 2$ .*

**Reduction Rule 3.** *Let  $C$  be a connected component of  $G - v$  for some vertex  $v \in G$ . If there exist  $a, b \in V(C)$  such that  $G - \{a, b\}$  is connected and there is an edge  $av$  but no edge  $bv$ , then mark  $a$  and  $b$ , delete them and set  $k' = k - 2$ .*

**Reduction Rule 4.** *Let  $C$  be a connected component of  $G - v$  for some vertex  $v \in G$ . If there exist  $a, b \in C$  such that  $G - \{a, b\}$  is connected and  $vabv$  is a positive triangle, then mark  $a$  and  $b$ , delete them and set  $k' = k - 4$ .*

**Reduction Rule 5.** *If there is a vertex  $v \in V(G)$  such that  $G - v$  has a connected component  $C$ ,  $G[V(C) \cup \{v\}]$  is a clique in  $G$ , and  $G[V(C) \cup \{v\}]$  does not contain a positive triangle, then delete  $C$  and set  $k' = k$ .*

**Reduction Rule 6.** *If  $a, b, c \in V(G)$ ,  $\{ab, bc\} \subseteq E(G)$  but  $ac \notin E(G)$ , and  $G - \{a, b, c\}$  is connected, then mark  $a, b, c$ , delete them and set  $k' = k - 1$ .*

**Reduction Rule 7.** *Let  $C, Y$  be the connected components of  $G - \{v, b\}$  for some vertices  $v, b \in V(G)$  such that  $vb \notin E(G)$ . If  $G[V(C) \cup \{v\}]$  and  $G[V(C) \cup \{b\}]$  are cliques not containing any positive triangles, then mark  $v$  and  $b$ , delete them, delete  $C$  and set  $k' = k - 1$ .*

**Definition 1.** *A one-way reduction rule is safe if it does not transform a No-instance into a YES-instance.*

The intuitive understanding of how a one-way reduction rule works is that it removes a portion of the graph (while decreasing the parameter from  $k$  to  $k'$ ) only if given any solution (i.e., a balanced subgraph) on the rest of the graph there is a way to extend it to the removed portion while always gaining an additional  $k - k'$  over the Poljak-Turzik bound.

**Lemma 3.** Let  $G$  be a connected graph. If  $C$  is a clique of  $G$  such that  $G - C$  is connected and if  $C$  contains a positive triangle, then either Rule 1 or Rule 2 applies.

*Proof.* Let  $abca$  be a positive triangle in  $C$ . Suppose Rule 1 does not apply. This means that  $G - \{a, b, c\}$  is not connected: more precisely,  $G - \{a, b, c\}$  has two components  $G - C$  and  $C - \{a, b, c\}$ . Note that  $C - \{a, b, c\}$  cannot contain a positive triangle, or otherwise Rule 1 would have applied. Therefore, Rule 2 applies.  $\square$

**Theorem 4.** [★] Rules 1-7 are safe.

We now show that the reduction rules preserve connectedness and that there is always one of them which applies to a graph with at least one edge. To show this, we use the following lemma, based on a result in [5] but first expressed in the following form in [3].

**Lemma 4.** [3] For any connected graph  $G$ , at least one of the following properties holds:

- A There exist  $v \in V(G)$  and  $X \subseteq V(G)$  such that  $G[X]$  is a connected component of  $G - v$  and  $G[X]$  is a clique;
- B There exist  $a, b, c \in V(G)$  such that  $G[\{a, b, c\}]$  is isomorphic to path  $P_3$  and  $G - \{a, b, c\}$  is connected;
- C There exist  $v, b \in V(G)$  such that  $vb \notin E(G)$ ,  $G - \{v, b\}$  is disconnected, and for all connected components  $G[X]$  of  $G - \{v, b\}$ , except possibly one,  $G[X \cup \{v\}]$  and  $G[X \cup \{b\}]$  are cliques.

**Lemma 5.** For a connected graph  $G$  with at least one edge, at least one of Rules 1-7 applies. In addition, the graph  $G'$  which is produced is connected.

*Proof.* It is not difficult to see that the graph  $G'$  is connected, since when it is not obvious, its connectedness is part of the conditions for the rule to apply.

If property A of Lemma 4 holds, and  $G[X]$  contains a positive triangle  $abca$ , then by Lemma 3 either Rule 1 or Rule 2 applies. If  $2 \leq |N_G(v) \cap X| \leq |X| - 1$ , then Rule 3 applies. If  $|N_G(v) \cap X| = |X|$  and there exist  $a, b \in X$  such that  $vabv$  is a positive triangle, Rule 4 applies; otherwise,  $G[X \cup \{v\}]$  contains no positive triangles, and Rule 5 applies. Finally, if  $N_G(v) \cap X = \{x\}$ , Rule 5 applies for  $x$  with clique  $G[X \setminus \{x\}]$ .

If property B of Lemma 4 holds, then Rule 6 applies. If property C of Lemma 4 holds, consider the case when  $G - \{v, b\}$  has two connected components. Let  $Z$  be the other connected component. If  $Z$  is connected to only one of  $v$  or  $b$ , then property A holds. Otherwise, if  $G[X \cup \{x\}]$  contains a positive triangle, where  $x \in \{v, b\}$ , then by Lemma 3 either Rule 1 or Rule 2 applies. So we may assume that  $G[X \cup \{b, v\}]$  contains no positive triangles, in which case Rule 7 applies.

If  $G - \{v, b\}$  has at least three connected components, at least two of them,  $X_1, X_2$ , form cliques with both  $v$  and  $b$  and possibly one component  $Y$  does not. Assume without loss of generality that  $Y$  has an edge to  $v$ . Then Rule 6 applies for the path  $x_1bx_2$ , where  $x_1 \in X_1, x_2 \in X_2$ .  $\square$

The following lemma gives structural results on  $S$  and  $G - S$ . Note that from now on,  $(G = (V, E), k)$  denotes the original instance of SIGNED MAX CUT ATLB and  $(G' = (V', E'), k')$  denotes the instance obtained by applying Rules 1-7 exhaustively. The set  $S \subseteq V$  denotes the set of vertices which are marked by the rules.

**Lemma 6.** *[\*] Given a connected graph  $G$ , if we apply Rules 1-7 exhaustively, either the set  $S$  of marked vertices has cardinality at most  $3k$ , or  $k' \leq 0$ . In addition,  $G - S$  is a forest of cliques that does not contain a positive triangle.*

Finally, it is possible to prove that SIGNED MAX CUT ATLB is FPT. First we state MAX-CUT-WITH-WEIGHTED-VERTICES as in [5].

**MAX-CUT-WITH-WEIGHTED-VERTICES**

*Instance:* A graph  $G$  with weight functions  $w_1 : V(G) \rightarrow \mathbb{N}_0$  and  $w_2 : V(G) \rightarrow \mathbb{N}_0$ , and an integer  $t \in \mathbb{N}$ .

*Question:* Does there exist an assignment  $f : V(G) \rightarrow \{1, 2\}$  such that  $\sum_{xy \in E} |f(x) - f(y)| + \sum_{f(x)=1} w_1(x) + \sum_{f(x)=2} w_2(x) \geq t$ ?

**Theorem 5.** SIGNED MAX CUT ATLB can be solved in time  $O^*(8^k)$ .

*Proof.* Let  $(G = (V, E), k)$  be an instance of SIGNED MAX CUT ATLB. Apply Rules 1-7 exhaustively, producing an instance  $(G' = (V', E'), k')$  and a set  $S \subseteq V$  of marked vertices. If  $k' \leq 0$ ,  $(G', k')$  is a trivial YES-instance. Since the rules are safe, it follows that  $(G, k)$  is a YES-instance, too.

Otherwise,  $k' > 0$ . Note that by Lemma 6,  $|S| \leq 3k$  and  $G - S$  is a forest of cliques, which is a chordal graph without positive triangles. Hence, by Corollary 3, we may assume that  $G - S$  does not contain positive edges.

Therefore, to solve SIGNED MAX CUT ATLB on  $G$ , we can guess a balanced subgraph of  $G[S]$ , induced by a partition  $(V_1, V_2)$ , and then solve MAX-CUT-WITH-WEIGHTED-VERTICES for  $G - S$ . The weight of a vertex  $v \in V(G - S)$  is defined in the following way: let  $n_i^+(v)$  be the number of positive neighbors of  $v$  in  $V_i$  and  $n_i^-(v)$  be the number of negative neighbors of  $v$  in  $V_i$ ; then  $w_1(v) = n_1^+(v) + n_2^-(v)$  and  $w_2(v) = n_2^+(v) + n_1^-(v)$ .

Since MAX-CUT-WITH-WEIGHTED-VERTICES is solvable in polynomial time on a forest of cliques (see Lemma 9 in [5]) and the number of possible partitions of  $S$  is bounded by  $2^{3k}$ , this gives an  $O^*(8^k)$ -algorithm to solve SIGNED MAX CUT ATLB.  $\square$

## 4 Kernelization

In this section, we show that SIGNED MAX CUT ATLB admits a kernel with  $O(k^3)$  vertices. The proof of Theorem 5 implies the following key result for our kernelization.

**Corollary 4.** *Let  $(G = (V, E), k)$  be an instance of SIGNED MAX CUT ATLB. In polynomial time, either we can conclude that  $(G, k)$  is a YES-instance or we can find a set  $S$  of at most  $3k$  vertices for which we may assume that  $G - S$  is a forest of cliques without positive edges.*

The kernel is obtained via the application of a new set of reduction rules and using structural results that bound the size of NO-instances  $(G, k)$ . First, we need some additional terminology. For a block  $C$  in  $G - S$ , let  $C_{\text{int}} = \{x \in V(C) : N_{G-S}(x) \subseteq V(C)\}$  be the *interior* of  $C$ , and let  $C_{\text{ext}} = V(C) \setminus C_{\text{int}}$  be the *exterior* of  $C$ . If a block  $C$  is such that  $C_{\text{int}} \cap N_G(S) \neq \emptyset$ ,  $C$  is a *special block*. We say a block  $C$  is a *path block* if  $|V(C)| = 2 = |C_{\text{ext}}|$ . A *path vertex* is a vertex which is contained only in path blocks. A block  $C$  in  $G - S$  is a *leaf block* if  $|C_{\text{ext}}| \leq 1$ .

The following reduction rules are two-way reduction rules: they apply to an instance  $(G, k)$  and produce an equivalent instance  $(G', k')$ .

**Reduction Rule 8.** *Let  $C$  be a block in  $G - S$ . If there exists  $X \subseteq C_{\text{int}}$  such that  $|X| > \frac{|V(C)| + |N_G(X) \cap S|}{2} \geq 1$ ,  $N_G^+(x) \cap S = N_G^+(X) \cap S$  and  $N_G^-(x) \cap S = N_G^-(X) \cap S$  for all  $x \in X$ , then delete two arbitrary vertices  $x_1, x_2 \in X$  and set  $k' = k$ .*

**Reduction Rule 9.** *Let  $C$  be a block in  $G - S$ . If  $|V(C)|$  is even and there exists  $X \subseteq C_{\text{int}}$  such that  $|X| = \frac{|V(C)|}{2}$  and  $N_G(X) \cap S = \emptyset$ , then delete a vertex  $x \in X$  and set  $k' = k - 1$ .*

**Reduction Rule 10.** *Let  $C$  be a block in  $G - S$  with vertex set  $\{x, y, u\}$ , such that  $N_G(u) = \{x, y\}$ . If the edge  $xy$  is a bridge in  $G - \{u\}$ , delete  $C$ , add a new vertex  $z$ , positive edges  $\{zv : v \in N_{G-u}^+(\{x, y\})\}$ , negative edges  $\{zv : v \in N_{G-u}^-(\{x, y\})\}$  and set  $k' = k$ . Otherwise, delete  $u$  and the edge  $xy$  and set  $k' = k - 1$ .*

**Reduction Rule 11.** *Let  $T$  be a connected component of  $G - S$  only adjacent to a vertex  $s \in S$ . Form a MAX-CUT-WITH-WEIGHTED-VERTICES instance on  $T$  by defining  $w_1(x) = 1$  if  $x \in N_G^+(s) \cap T$  ( $w_1(x) = 0$  otherwise) and  $w_2(y) = 1$  if  $y \in N_G^-(s) \cap T$  ( $w_2(y) = 0$  otherwise). Let  $\beta(G[V(T) \cup \{s\}]) = \text{pt}(G[V(T) \cup \{s\}]) + \frac{p}{4}$ . Then delete  $T$  and set  $k' = k - p$ .*

Note that the value of  $p$  in Rule 11 can be found in polynomial time by solving MAX-CUT-WITH-WEIGHTED-VERTICES on  $T$ .

A two-way reduction rule is *valid* if it transforms YES-instances into YES-instances and NO-instances into NO-instances. Theorem 6 shows that Rules 8-11 are valid.

**Theorem 6.** *[ $\star$ ] Rules 8-11 are valid.*

To show the existence of a kernel with  $O(k^3)$  vertices, it is enough to give a bound on the number of non-path blocks, the number of vertices in these blocks and the number of path vertices. This is done by Corollaries 6 and 7 and Lemma 11.

While Lemma 11 applies to any graph reduced by Rule 8, the proofs of Corollaries 6 and 7 rely on Lemma 10, which gives a general structural result on forest of cliques with a bounded number of special blocks and bounded path length. Corollary 5 and Lemma 9 provide sufficient conditions for a reduced instance to be a YES-instance, thus producing a bound on the number of special blocks and the path length of NO-instances. Lastly, Theorem 7 puts the results together to show the existence of the kernel.

Henceforth, we assume that the instance  $(G, k)$  is such that  $G$  is reduced by Rules 8-11,  $G - S$  is a forest of cliques which does not contain a positive edge and  $|S| \leq 3k$ .

**Lemma 7.** Let  $T$  be a connected component of  $G - S$ . Then for every leaf block  $C$  of  $T$ ,  $N_G(C_{\text{int}}) \cap S \neq \emptyset$ . Furthermore, if  $|N_G(S) \cap V(T)| = 1$ , then  $T$  consists of a single vertex.

*Proof.* We start by proving the first claim. Note that if  $T = C$  consists of a single vertex, then  $N_G(C_{\text{int}}) \cap S \neq \emptyset$  since  $G$  is connected. So assume that  $C$  has at least two vertices. Suppose that  $N_G(C_{\text{int}}) \cap S = \emptyset$  and let  $X = C_{\text{int}}$ . Then if  $|C_{\text{int}}| > |C_{\text{ext}}|$ , Rule 8 applies. If  $|C_{\text{int}}| = |C_{\text{ext}}|$  then Rule 9 applies. Otherwise,  $|C_{\text{int}}| < |C_{\text{ext}}|$  and since  $|C_{\text{ext}}| \leq 1$  (as  $C$  is a leaf block),  $C$  has only one vertex, which contradicts our assumption above. For the second claim, first note that since  $|N_G(S) \cap V(T)| = 1$ ,  $T$  has one leaf block and so  $T$  consists of a single block. Let  $N_G(S) \cap V(T) = \{v\}$  and  $X = V(T) - \{v\}$ . If  $|X| > 1$ , Rule 8 applies. If  $|X| = 1$ , Rule 9 applies. Hence  $V(T) = \{v\}$ .  $\square$

Let  $\mathcal{B}$  be the set of non-path blocks.

**Lemma 8.** If there exists a vertex  $s \in S$  such that  $\sum_{C \in \mathcal{B}} |N_G(C_{\text{int}}) \cap \{s\}| \geq 2(|S| - 1 + k)$ , then  $(G, k)$  is a YES-instance.

*Proof.* Form  $T \subseteq N_G(s)$  by picking a vertex from each block  $C$  for which  $|N_G(C_{\text{int}}) \cap \{s\}| = 1$ : if there exists a vertex  $x \in C_{\text{int}}$  such that  $N_G(x) \cap S = \{s\}$ , pick this, otherwise pick  $x \in C_{\text{int}}$  arbitrarily. Let  $U = T \cup \{s\}$  and  $W = V \setminus U$ .

Observe that  $G[U]$  is balanced by Theorem 1 as  $G[U]$  is a tree. Thus  $\beta(G[U]) = |T| = \frac{|T|}{2} + \frac{|T|}{4} + \frac{|T|}{4} = \text{pt}(G[U]) + \frac{|T|}{4}$ .

Consider a connected component  $Q$  of  $G - S$ . By Rule 11,  $|N_G(Q) \cap S| \geq 2$  and by Lemma 7, if  $|N_G(S) \cap V(Q)| = 1$  then  $Q$  consists of a single vertex. Otherwise, either  $(N_G(S) \setminus N_G(s)) \cap V(Q) \neq \emptyset$ , or  $Q$  has at least two vertices in  $T$ . Moreover, note that the removal of interior vertices does not disconnect the component itself. Hence  $G[W]$  has at most  $(|S| - 1) + \frac{|T|}{2}$  connected components. Applying Lemma 1,  $\beta(G) \geq \text{pt}(G) + \frac{|T|}{4} - \frac{(|S|-1)+\frac{|T|}{2}}{4} = \text{pt}(G) + \frac{|T|}{8} - \frac{|S|-1}{4}$ . Hence if  $|T| \geq 2(|S| - 1 + k)$ , then  $(G, k)$  is a YES-instance.  $\square$

**Corollary 5.** [\*] If  $\sum_{C \in \mathcal{B}} |N_G(C_{\text{int}}) \cap S| \geq |S|(2|S| - 3 + 2k) + 1$ , the instance is a YES-instance. Otherwise,  $\sum_{C \in \mathcal{B}} |N_G(C_{\text{int}}) \cap S| \leq 3k(8k - 3)$ .

**Lemma 9.** [\*] If in  $G - S$  there exist vertices  $U = \{u_1, u_2, \dots, u_p\}$  such that  $N_{G-S}(u_i) = \{u_{i-1}, u_{i+1}\}$  for  $2 \leq i \leq p - 1$ , and  $p \geq |S| + k + 1$ , then  $(G, k)$  is YES-instance. Otherwise,  $p \leq 4k$ .

In  $G - S$ , a *pure path* is a path consisting exclusively of path vertices. Note that every path vertex belongs to a unique pure path.

**Lemma 10.** [\*] Suppose  $G - S$  has at most  $l$  special blocks and the number of vertices in each pure path is bounded by  $p$ . Then  $G - S$  contains at most  $2l$  non-path blocks and  $2pl$  path vertices.

**Corollary 6.**  $G - S$  contains at most  $6k(8k - 3)$  non-path blocks and  $24k^2(8k - 3)$  path vertices.

*Proof.* By Corollary 5,  $G - S$  contains at most  $3k(8k - 3)$  special blocks and by Lemma 9, the length of every pure path is bounded by  $4k$ . Thus, Lemma 10 implies that  $G - S$  contains at most  $6k(8k - 3)$  non-path blocks and  $24k^2(8k - 3)$  path vertices.  $\square$

**Corollary 7.** *[\*]  $G - S$  contains at most  $12k(8k - 3)$  vertices in the exteriors of non-path blocks.*

**Lemma 11.** *[\*] For a block  $C$ , if  $|V(C)| \geq 2|C_{\text{ext}}| + |N_G(C_{\text{int}}) \cap S|(2|S| + 2k + 1)$ , then  $(G, k)$  is a YES-instance. Otherwise,  $|V(C)| \leq 2|C_{\text{ext}}| + |N_G(C_{\text{int}}) \cap S|(8k + 1)$ .*

**Theorem 7.** *SIGNED MAX CUT ATLB has a kernel with  $O(k^3)$  vertices.*

*Proof.* Let  $(G = (V, E), k)$  be an instance of SIGNED MAX CUT ATLB. As in Theorem 5, apply Rules 1-7 exhaustively: either the instance is a YES-instance, or there exists  $S \subseteq V$  such that  $|S| \leq 3k$  and  $G - S$  is a forest of cliques which does not contain a positive edge.

Now, apply Rules 8-11 exhaustively to  $(G, k)$  to obtain a new instance  $(G', k')$ . If  $k' \leq 0$ , then  $(G, k)$  is a YES-instance since Rules 8-11 are valid. Now let  $G = G'$ ,  $k = k'$ . Check whether  $(G, k)$  is a YES-instance due to Corollary 5, Lemma 9 or Lemma 11. If this is not the case, by Corollary 6,  $G - S$  contains at most  $6k(8k - 3)$  non-path blocks and  $24k^2(8k - 3)$  path vertices. Hence, by Lemma 11,  $|V(G)|$  is at most

$$|S| + 24k^2(8k - 3) + \sum_{C \in \mathcal{B}} |V(C)| \leq O(k^3) + 2 \sum_{C \in \mathcal{B}} |C_{\text{ext}}| + (8k + 1) \sum_{C \in \mathcal{B}} |N_G(C_{\text{int}}) \cap S|$$

Now, applying Corollary 5 and Corollary 7, we obtain:

$$|V(G)| \leq O(k^3) + 48k(8k - 3) + 3k(8k - 3)(8k + 1) = O(k^3).$$

$\square$

It is not hard to verify that no reduction rule of this paper increases the number of positive edges. Thus, considering an input  $G$  of MAX CUT ATLB as an input of SIGNED MAX CUT ATLB by assigning minus to each edge of  $G$ , we have the following:

**Corollary 8.** *MAX CUT ATLB has a kernel with  $O(k^3)$  vertices.*

## 5 Extensions and Open Questions

In the previous sections, the input of SIGNED MAX CUT ATLB is a signed graph without parallel edges. However, in some applications (cf. [9,10]), signed graphs may have parallel edges of opposite signs. We may easily extend inputs of SIGNED MAX CUT ATLB to such graphs. Indeed, if  $G$  is such a graph we may remove all pairs of parallel edges from  $G$  and obtain an equivalent instance of SIGNED MAX CUT ATLB.

In fact, the Poljak-Turzík bound can be extended to edge-weighted graphs [15]. Let  $G$  be a connected signed graph in which each edge  $e$  is assigned a positive weight  $w(e)$ . The weight  $w(Q)$  of an edge-weighted graph  $Q$  is the sum of weights of its edges. Then  $G$  contains a balanced subgraph with weight at least  $w(G)/2 + w(T)/4$ , where

$T$  is a spanning tree of  $G$  of minimum weight [15]. It would be interesting to establish parameterized complexities of MAX CUT ATLB and SIGNED MAX CUT ATLB extended to edge-weighted graphs using the Poljak-Turzík bound above.

**Acknowledgement.** We are thankful to Fedor Fomin for a useful discussion.

## References

1. Chiang, C., Kahng, A.B., Sinha, S., Xu, X., Zelikovsky, A.Z.: Fast and efficient bright-field AAPSM conflict detection and correction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26(1), 11–126 (2007)
2. Crowston, R., Fellows, M., Gutin, G., Jones, M., Rosamond, F., Thomassé, S., Yeo, A.: Simultaneously Satisfying Linear Equations Over  $\mathbb{F}_2$ : MaxLin2 and Max- $r$ -Lin2 Parameterized Above Average. In: FSTTCS 2011. LIPICS, vol. 13, pp. 229–240 (2011)
3. Crowston, R., Gutin, G., Jones, M.: Directed Acyclic Subgraph Problem Parameterized above the Poljak-Turzík Bound. In: FSTTCS 2012. LIPICS, vol. 18, pp. 400–411 (2012)
4. Crowston, R., Gutin, G., Jones, M., Muciaccia, G.: Maximum Balanced Subgraph Problem Parameterized Above Lower Bound. arXiv:1212.6848
5. Crowston, R., Jones, M., Mnich, M.: Max-Cut Parameterized above the Edwards-Erdős Bound. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part I. LNCS, vol. 7391, pp. 242–253. Springer, Heidelberg (2012)
6. DasGupta, B., Enciso, G.A., Sontag, E.D., Zhang, Y.: Algorithmic and complexity results for decompositions of biological networks into monotone subsystems. *Biosystems* 90(1), 161–178 (2007)
7. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer (1999)
8. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer (2006)
9. Gülpınar, N., Gutin, G., Mitra, G., Zverovitch, A.: Extracting Pure Network Submatrices in Linear Programs Using Signed Graphs. *Discrete Applied Mathematics* 137, 359–372 (2004)
10. Gutin, G., Zverovitch, A.: Extracting pure network submatrices in linear programs using signed graphs, Part 2. *Communications of DQM* 6, 58–65 (2003)
11. Harary, F.: On the notion of balance of a signed graph. *Michigan Math. J.* 2(2), 143–146 (1953)
12. Hüffner, F., Betzler, N., Niedermeier, R.: Optimal edge deletions for signed graph balancing. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 297–310. Springer, Heidelberg (2007)
13. Mnich, M., Philip, G., Saurabh, S., Suchý, O.: Beyond Max-Cut:  $\lambda$ -Extendible Properties Parameterized Above the Poljak-Turzík Bound. In: FSTTCS 2012. LIPICS, vol. 18, pp. 412–423 (2012)
14. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms, Oxford UP (2006)
15. Poljak, S., Turzík, D.: A polynomial time heuristic for certain subgraph optimization problems with guaranteed worst case bound. *Discrete Mathematics* 58(1), 99–104 (1986)
16. Raman, V., Saurabh, S.: Parameterized algorithms for feedback set problems and their duals in tournaments. *Theor. Comput. Sci.* 351(3), 446–458 (2006)
17. Zaslavsky, T.: Bibliography of signed and gain graphs. *Electronic Journal of Combinatorics*, DS8 (1998)

# A Toolbox for Provably Optimal Multistage Strict Group Testing Strategies

Peter Damaschke and Azam Sheikh Muhammad

Department of Computer Science and Engineering  
Chalmers University, 41296 Göteborg, Sweden  
`{ptr,azams}@chalmers.se`

**Abstract.** Group testing is the problem of identifying up to  $d$  defectives in a set of  $n$  elements by testing subsets for the presence of defectives. Let  $t(n, d, s)$  be the optimal number of tests needed by an  $s$ -stage strategy in the strict group testing model where the searcher must also verify that no more than  $d$  defectives are present. We develop combinatorial tools that are powerful enough to compute many exact  $t(n, d, s)$  values. This extends the work of Huang and Hwang (2001) for  $s = 1$  to multistage strategies. The latter are interesting since it is known that asymptotically nearly optimal group testing is possible already in  $s = 2$  stages. Besides other tools we generalize  $d$ -disjunct matrices to any candidate hypergraphs, which enables us to express optimal test numbers for  $s = 2$  as chromatic numbers of certain conflict graphs. As a proof of concept we determine almost all test numbers for  $n \leq 10$ , and  $t(n, 2, 2)$  for some larger  $n$ .

## 1 Introduction

In the *group testing* problem, a set of  $n$  elements is given, each being either *defective (positive)* or *non-defective (negative)*. Let  $P$  denote the unknown set of positive elements. A *group test* takes any subset  $Q$  of elements, called a *pool*. The test (or pool) is positive if  $Q \cap P \neq \emptyset$ , and negative otherwise. In the latter case, obviously, all elements in  $Q$  are recognized as negative. The goal is to identify  $P$  using few tests. A group testing strategy may be organized in  $s$  *stages*, where all tests within a stage are executed in parallel. In *adaptive* group testing  $s$  is not limited, hence tests can be done sequentially. Case  $s = 1$  is called *nonadaptive*. Small  $s$  are desired in applications where the tests take much time.

It is expected that  $|P| \leq d$ , for some fixed bound  $d$ , with the understanding that  $|P| > d$  is unlikely but not impossible. A searcher wants to identify  $P$  if  $|P| \leq d$ , and just report “ $|P| > d$ ” otherwise. This setting is called *strict group testing*, in contrast to *hypergeometric group testing* where  $|P| \leq d$  is “promised” to the searcher. It was argued in, e.g., [1] that strict group testing is preferable. It does not rely on the assumption  $|P| \leq d$ .

In a few lines one cannot possibly give even a cursory overview of the applications of group testing in biology and computer science, and of the main complexity results. We only refer to the books [7,8] and a few recent papers [2,5,11,16,19].

as entry points to further studies. As is well known [9],  $O(d \log n)$  tests are not enough if  $s = 1$ . The breakthrough result in [6] showed that  $O(d \log n)$  tests are sufficient already if  $s = 2$ , followed by improvements of the hidden constant factor [10,3]. However, such asymptotic results are designed for optimal behavior as  $n$  goes to infinity, but even asymptotically optimal strategies do not necessarily entail the optimal strategy for specific input sizes  $n$ . Another motivation of the quest for optimal strategies for specific  $n$  is that the pool sizes of asymptotically optimal strategies increase with  $n$ , but in some applications, large pools may be infeasible (because of technical obstacles, chemical dilution, etc.). Still we can split an instance into many small instances and solve them independently, now each with optimal efficiency. To mention a practical example, screening blood donations for various infectious diseases is performed at some labs in instances (“minipools”) of, e.g., 16 samples [18], and group testing is proposed [20] to reduce the waiting times (can be days) and the considerable costs (for millions of donations annually).

Due to the preceding discussion, we define  $t(n, d, s)$  to be the optimal worst-case number of tests needed by a strict group testing strategy for  $n$  elements, up to  $d$  defectives, and  $s$  stages. Some *monotonicity* relations hold trivially: If  $n \leq n'$ ,  $d \leq d'$ , and  $s \geq s'$  then  $t(n, d, s) \leq t(n', d', s')$ . If  $t(n, d, s) = t(n, d, n)$ , we write  $t(n, d, s+)$  to indicate that more stages would not lower the test number.

The work closest to ours is [15] from which the exact  $t(n, d, 1)$  values follow for all  $n \leq 14$  and arbitrary  $d$ , as well as some test numbers for larger  $n$ . The novelty of the present work is that we extend the scope to  $s > 1$  stages, which saves tests compared to  $s = 1$ . To our best knowledge, this terrain has not been systematically explored, with a few exceptions:  $t(n, 1, 1) = \log_2 n + o(\log_2 n)$  is the smallest  $k$  with  $\binom{k}{k/2} \geq n$  due to [17],  $t(n, 1, 2+) = \lceil \log_2 n \rceil + 1$  is a side result in [4], and [12] gives partial results on adaptive strategies. The focus in [12,15] is on the question for which  $n, d$  group testing saves tests, compared to trivial individual testing.

While an upper bound on a specific  $t(n, d, s)$  is established once a strategy is found, the main challenge is to prove matching lower bounds. We stress that this cannot be done naively by considering all possible pooling designs in every stage, as their number is doubly exponential in  $n$ . Instead we have developed several combinatorial tools. Each of our lemmas viewed in isolation is relatively simple, but *together they enable us to determine many  $t(n, d, s)$  values exactly*, by a branch-and-bound approach. Yet they should also be of independent interest as structural results. Due to severe space limitations, this paper version demonstrates the use of the theory only for  $n \leq 10$  (and all  $d, s$ ) and for  $t(n, 2, 2)$ , although even more  $t(n, d, s)$  could be managed with our current techniques.

## 2 Notation

A  $k$ -set ( $\leq k$ -set,  $\geq k$ -set) is a set with exactly (at most, at least)  $k$  elements. We use this notation also for pools, hyperedges, and other sets with special roles. A *hypergraph* is a set of *vertices* equipped with a family of subsets called *hyperedges*.

A hypergraph with only  $\leq 2$ -hyperedges is a *graph* with *edges*. A 1-hyperedge is a *loop*. Note that we allow *parallel* hyperedges, that is, hyperedges being identical as sets may occur multiple times. Two sets are *incomparable* if neither is a subset of the other one. A family of pairwise incomparable sets is an *antichain*. We use standard graph-theoretic symbols:  $K_n$ ,  $C_n$ ,  $K_{m,n}$  is the clique, cycle, and complete bipartite graph, respectively, with the indicated vertex numbers. A *forest* (union of trees) is a cycle-free graph, and a *leaf* is a vertex incident to only 1 edge.

During the course of applying a group testing strategy, an element which has not appeared in any negative pool so far is called a *candidate element*. A *candidate set* is a set of up to  $d$  candidate elements that is consistent with all previous test outcomes. That is, any candidate set is possibly the true set  $P$ . The name candidate element reflects the searcher's knowledge: An element is possibly positive until it is discarded, as the member of some negative pool. Therefore we can have candidate elements outside all candidate sets, called *dummy elements*. For example, if  $n = 5$  and  $d = 2$ , and we test 2 disjoint 2-pools with positive outcome, then the 5th element was in no negative pool so far, but any candidate 2-set must take one element from both pools.

In the *candidate hypergraph*, the candidate elements are the vertices and the candidate sets form the hyperedges. For  $d = 2$ , the candidate hypergraph is just a *candidate graph*, possibly with loops. From the definitions it follows that an instance of the strict group testing problem is solved if and only if the candidate hypergraph has exactly one hyperedge and no dummy elements.

A *pool hypergraph* represents a set of pools in dual form: Vertices  $p_1, p_2, p_3, \dots$  are the pools, and hyperedges are the candidate elements. A vertex belongs to a hyperedge if the corresponding pool contains the corresponding element. Dummy elements are represented as loops at a symbolic *null vertex*  $p_0$ .

### 3 Lower-Bound Tools

The simple *counting bound* says that the number of tests is at least  $\log_2$  of number of outcomes. In particular we have  $t(n, d, n) \geq \log_2(1 + \sum_{i=0}^d \binom{n}{i})$  where the summand 1 accounts for the outcome " $|P| > d$ ". First we give a more powerful lower bound, which often guarantees one more test.

**Lemma 1.** *If  $m > 2^r$  candidate sets exist which form an antichain, then strict group testing requires at least  $r + 2$  tests, even adaptively.*

*Proof.* It suffices to consider  $m = 2^r + 1$ . We use induction on  $r$ . We first prove that at least 2 tests are required for the base case  $r = 0$ , that is,  $m = 2$ . Let  $C$  and  $C'$  be incomparable candidate sets. Assume that one pool is enough. If it intersects either none or both of  $C$  and  $C'$ , it cannot distinguish between these candidate sets. Thus the pool must intersect  $C$  and be disjoint to  $C'$  (or vice versa). Now a positive outcome leaves the searcher unsure about the status (defective or not) of the elements in  $C' \setminus C$ . For the inductive step assume that the claim is true for  $r$ , and let  $m = 2^{r+1} + 1$ . In one test outcome, the majority of

candidate sets remain. Therefore, after the first test we would keep at least  $2^r + 1$  candidate sets. Using the inductive hypothesis the claim holds for  $r + 1$ .  $\square$

**Examples.** For  $d = 2$  we list, for some pool hypergraphs, the test numbers enforced by Lemma 1 if all pools respond positively. Note that candidate sets of the same size form an antichain. These test numbers will be used to get our specific  $t(n, d, s)$  results.

- 2 loops at a vertex: 2 candidate 1-sets, 2 tests.
- 2 loops at a pool vertex and 1 loop at  $p_0$ : 3 candidate 2-sets, 2 tests.
- 3 loops at a vertex: 3 candidate 2-sets, 3 tests.
- 4 loops at a vertex: 6 candidate 2-sets, 4 tests.
- 2 loops at distance 1 or 2: 3 candidate sets, 3 tests.
- $C_3$ : 3 candidate 2-sets, 3 tests.
- $C_4$ : 2 candidate 2-sets, 2 tests.

The following lemmas are evident monotonicity observations (proved by a “the searcher gets less information”-argument), but they are extremely useful for limiting the strategies that must be considered in lower-bound proofs. Recall that we consider strict group testing in a prescribed number  $s$  of stages.

**Lemma 2.** *In response to a given deterministic test strategy, consider a test answering strategy  $A$  that enforces  $t$  tests in the worst case. If the searcher replaces some pool  $Q$ , that is negative (positive) in  $A$ , with a subset (superset) of  $Q$ , then still at least  $t$  tests are needed in the worst case.*  $\square$

**Lemma 3.** *Suppose that the outcomes of some pools of a stage are revealed to the searcher, and then she may redesign her other pools from scratch. If  $t$  further tests are not sufficient despite redesign, then they are not sufficient for the original problem instance either.*  $\square$

Pools can be arranged as a Boolean matrix. It is well known that a pooling design solves *strict* group testing for  $s = 1$  if and only if it forms a  $d$ -disjunct matrix. We withhold the definition of  $d$ -disjunct, as we present a straight generalization to arbitrary candidate hypergraphs. (This should not be confused with the concept in [13] which addresses a different group testing problem.)

**Theorem 1.** *A nonadaptive strategy solves strict group testing for a given candidate hypergraph if and only if, for every pair of a candidate set  $C$  and a candidate element  $v \notin C$ , it has a pool  $Q$  such that  $v \in Q$  and  $C \cap Q = \emptyset$ .*

*Proof.* To prove necessity, let  $C$  be a candidate set and  $v \notin C$  a candidate element. If  $C = P$  then  $v$  must be recognized as negative. Hence some negative pool must contain  $v$ , in particular, this pool must be disjoint to  $C$ . Next we prove sufficiency. If  $P = C$  then clearly all elements outside  $C$  will be recognized as negative, and all elements in  $P$  are still candidate elements. Suppose  $D \subset P$  is also a candidate set. Then the searcher can falsify the assumption  $P = D$ , since then all elements in  $C \setminus D$  would be negative and be recognized as such, too. Hence  $P$  is the only remaining candidate set, and no dummy elements remain. The case when  $P$  is none of the candidate sets (but  $|P| > d$  instead) is also easily recognized by the fact that more than  $d$  candidate elements are retained.  $\square$

A *partial vector* is a vector where any position either has a *fixed* value 0 or 1, or remains *open*, indicated by the \* symbol. We index the candidate elements by  $i = 1, 2, 3, \dots$ , and we encode every pair of a candidate set  $C$  and a candidate element  $v \notin C$  as a partial vector as follows. We assign 0 to all positions of elements of  $C$ , and 1 to the position of  $v$ . All other positions are open. Two partial vectors *conflict* if one has 0 while the other one has 1 at the same position. Two partial vectors that do not conflict are *compatible*. We translate the candidate hypergraph into a *conflict graph* defined as follows. The vertices represent the partial vectors for all  $C$  and  $v \notin C$ , and two vertices are adjacent if the corresponding partial vectors are in conflict. A pool is naturally represented as its indicator vector, that is, a bit vector with 1 at all positions of elements in the pool, and 0 elsewhere. A pool is said to *cover* a partial vector if all fixed positions in the partial vector have the same value, 1 or 0, as in the pool's indicator vector. The smallest number of colors needed to color a graph  $G$ , such that adjacent vertices get distinct colors, is known as chromatic number  $\chi(G)$ . We refer to  $\chi(G)$  of a conflict graph  $G$  as *conflict chromatic number*.

**Theorem 2.** *Solving the strict group testing problem nonadaptively for a given candidate hypergraph is equivalent to coloring the conflict graph. Consequently, the conflict chromatic number equals the number of tests required.*

*Proof.* The condition in Theorem 1 can be equivalently expressed saying that, for every pair of a candidate set  $C$  and candidate element  $v \notin C$ , some pool must cover its partial vector. Observe that a single pool can cover a set of partial vectors if and only if they are pairwise compatible. In the conflict graph, compatible partial vectors are represented by nonadjacent vertices. Thus, a subset of partial vectors can be covered by a single pool if and only if the vertices form an independent set. Since partitioning a graph into a minimum number of independent sets is equivalent to graph coloring, the assertion follows.  $\square$

Of course, the fact that graph coloring is NP-hard does not stop us from computing  $\chi(G)$  for specific conflict graphs  $G$  needed for our purposes. As a first example, the candidate graph  $C_4 = K_{2,2}$  has conflict chromatic number 4, since the partial vectors  $[010*]$ ,  $[0*10]$ ,  $[*001]$ ,  $[10*0]$  conflict pairwise. Therefore, if  $d = 2$ ,  $s = 2$ , and the pool graph used in stage 1 has 2 vertices with 2 loops each, then at least 4 more tests are required in stage 2.

A nonadaptive strategy on the candidate graph  $K_{1,k}$  requires  $t(k, 1, 1)$  tests, as the central vertex together with either leaf is a candidate set. For instance, if  $d = 2$ ,  $s = 2$ , and the pool graph used in stage 1 has an edge  $p_1p_2$  and a total of  $k$  loops at  $p_1$ ,  $p_2$ , and  $p_0$ , then we get this situation. Now let  $K_{1,k} + e$  denote the  $k$ -star  $K_{1,k}$  with one extra edge between two of the leaves. We can prove the necessity of 1 more test using the conflict chromatic number:

**Lemma 4.** *A nonadaptive strict group testing strategy for the candidate graph  $K_{1,k} + e$  requires  $t(k, 1, 1) + 1$  tests.*

*Proof.* Assume that  $t(k, 1, 1)$  tests are enough. Leaving aside the candidate 2-set represented by the extra edge  $e$ , we first consider the partial vectors due to the  $k$  edges of the  $k$ -star. (See definitions prior to Theorem 2.) Let the first position correspond to the center of the  $k$ -star, while the further  $k$  positions correspond to the leaves. All partial vectors from the  $k$ -star have the form  $[0 \dots]$ , since all edges share the center vertex. The second defective is either of the  $k$  leaves. Thus we need already  $t(k, 1, 1)$  pools to cover (or “color”) all these partial vectors. Without loss of generality let  $e$  be the edge between the 2nd and 3rd vertex. Among the partial vectors due to  $e$ , we have in particular  $[100 * \dots]$ . This conflicts all earlier partial vectors with 0 at the 1st position. Thus, another color (i.e., pool) is needed to cover this partial vector.  $\square$

**Example.** Let  $n = 4$ ,  $d = 2$ ,  $s = 2$ , let the pool graph in stage 1 be  $C_3$  with 1 loop at 1 vertex, and these pools are positive. Then the candidate graph is  $K_{1,k} + e$ , hence  $t(3, 1, 1) + 1 = 4$  tests are needed in stage 2.

Let  $G_1$  and  $G_2$  be any two candidate hypergraphs on disjoint sets of vertices (elements). We define their *product*  $G_1 \times G_2$  as the candidate hypergraph whose vertices are all elements from  $G_1$  and  $G_2$ , with the hyperedges  $e_1 \cup e_2$  for any pair of hyperedges  $e_1$  from  $G_1$ , and  $e_2$  from  $G_2$ . Let  $t_i$  be the optimal number of tests for nonadaptive strict group testing when  $G_i$  is given ( $i = 1, 2$ ), that is, the respective conflict chromatic number. Trivially,  $G_1 \times G_2$  needs at most  $t_1 + t_2$  tests, since we may consider  $G_1 \times G_2$  as two independent “parallel” problem instances. A natural conjecture is that  $t_1 + t_2$  is also optimal. However the difficulty is that the searcher is free to use pools intersecting both vertex sets, which may cleverly save some tests. In fact, we are able to prove the conjecture in special cases only, yet they are powerful for our purposes. The prototype is the following case that uses similar reasoning as in Lemma 4.

**Lemma 5.** *Let  $G_1$  be the candidate graph with hyperedges  $\{v_1\}$  and  $\{v_2\}$  (that is, exactly one of these 2 elements is defective), and  $G_2$  arbitrary. Then nonadaptive strict group testing on  $G_1 \times G_2$  needs  $t_2 + 2$  tests.*

*Proof.* Let  $v_3, v_4, \dots$  denote the elements of  $G_2$ . Pools must cover all partial vectors according to Lemma 1 and Theorem 2. First consider the candidate sets of  $G_1 \times G_2$  where  $v_1$  is positive. Their partial vectors have the fixed value 0 at the 1st position. Hence  $t_2$  pools are needed to cover already these partial vectors. Assume that  $t_2 + 1$  pools are sufficient for  $G_1 \times G_2$ . Every partial vector of the form  $[10 \dots]$ , with further 0s at the positions of some candidate set from  $G_2$ , conflict with the  $t_2$  former pools. Hence all partial vectors  $[10 \dots]$  must be covered by the same last pool. Since all  $v_i$ ,  $i > 2$ , are candidate elements in  $G_2$  and give rise to 0s, it follows that this last pool can only be  $\{v_1\}$ . The key step is that, by symmetry, the pool  $\{v_2\}$  must also exist. Now the indicator vectors of these pools,  $[100 \dots 0]$  and  $[010 \dots 0]$ , conflict with all  $t_2$  pools needed to cover the partial vectors from  $G_2$ , since each of these has fixed value 1 at some position  $v_i$ ,  $i > 2$ . In total we need  $t_2 + 2$  pools.  $\square$

We can similarly prove a more general version of Lemma 5, however with a rather technical condition to  $G_1$ . We omit this elaboration. In the reported examples we will only apply the above basic case. One consequence that we use is that the candidate graph  $K_{2,n}$  needs  $t(n, 1, 1) + 2$  nonadaptive tests. The power of Lemma 5 is also illustrated by the following example: Consider the product of  $k$  copies of the above  $G_1$ , that is,  $k$  disjoint pairs of elements, where one element of each pair is defective. Since  $2^k$  candidate  $k$ -sets exist, the counting bound is only  $k$  tests, whereas inductive application of Lemma 5 gives the tight lower bound  $2k$ .

## 4 Upper-Bound Tools (Sub-Strategies)

**Lemma 6.** *Any candidate hypergraph of  $m$  candidate sets permits a nonadaptive strict group testing strategy with at most  $m$  tests.*

*Proof.* Test all complements of candidate sets. These  $m$  pools completely adhere to Theorem 1: For every candidate set  $C$  and candidate element  $v \notin C$ , the complement of  $C$  includes  $v$  and is disjoint to  $C$ . The assertion follows.  $\square$

**Lemma 7.** *Let  $G$  be a candidate hypergraph with  $n > d$  vertices where all candidate sets are  $d$ -sets. Let  $G'$  be obtained from  $G$  by adding dummy elements. Then the optimal test number on  $G'$  (for unchanged  $d, s$ ) is the same as for  $G$ .*

*Proof.* Add the dummy elements to every pool of an optimal strategy for  $G$ . Since  $n - d > 0$  elements must be discarded, in every possible application of the strategy (i.e., for any test outcomes), at least one pool is negative, or we recognize  $|P| > d$ . This negative pool also discards the dummy elements.  $\square$

For instance, candidate graph  $K_{1,3} + e$  can be solved with 3 tests when we allow 2 stages: Let the candidate 2-sets be the edges  $v_1v_2$ ,  $v_1v_3$ ,  $v_1v_4$ ,  $v_2v_3$ . We only test  $\{v_2\}$  in stage 1. Either positive or negative, there remain 2 candidate 2-sets for stage 2 and perhaps dummy elements, which requires by Lemma 6 and Lemma 7 only 2 more tests in stage 2.

## 5 Optimal Strategies for Small Instances

Our aim is now to get exact values  $t(n, d, s)$  for as many feasible combinations of  $n, d, s$  as possible. Recall that the  $t(n, 1, 1)$ , and  $t(n, 1, 2+)$  are already completely known, thus we study  $d \geq 2$  only. The methodology can be summarized as follows. In the pool (hyper)graphs in stage 1 we identify certain subsets  $W$  of vertices. If all pools in  $W$  are positive and the others negative, the candidate elements are precisely the edges in the subgraph induced by  $W$ , and the defective edges must cover  $W$ . Then we show that the resulting candidate (hyper)graphs enforce too many further tests, by our lower-bound techniques. – This “practical” section is intended to be a proof of concept. Readers may skip any items without losing their thread. Strategy descriptions are highlighted by (S).

$t(3, 2, 1+) = 3, t(4, 2, 1+) = 4, t(5, 2, 1+) = 5$  hold by the counting bound. For  $n = 6$  the counting bound gives only  $t(6, 2, 1+) \geq 5$ , nevertheless we can prove:

**$t(6, 2, 1+) = 6$ .** It suffices to consider adaptive strategies. If we begin with a 1-pool and it is negative, we need  $t(5, 2, 1+) = 5$  further tests. If we begin with a 2-pool and it is positive, there remain  $9 > 2^3$  candidate 2-sets, hence Lemma 1 requires 5 more tests. Due to Lemma 2 we need not consider more cases.

**$t(7, 2, 3+) = 6$ .** The lower bound follows from  $t(6, 2, 3) = 6$ . (**S**) For the upper bound, use 3 mutually disjoint 2-pools in stage 1. If at most 1 of them responds positive, at most 3 candidate elements are left, that can be tested individually in stage 2. If all 3 of them respond positive, we conclude  $|P| > 2$ . If exactly 2 of them respond positive, then, in stage 2, we query separately 1 element from each positive pool (2 tests). A negative outcome means the other element is positive, whereas a positive outcome renders the queried element positive. Thus, one positive element is recognized in both cases. A 6th test in stage 3 on the remaining candidates confirms they are negative, or yields  $|P| > 2$ .

**$t(7, 2, 2) = 7$ .** We assume for contradiction that  $t(7, 2, 2) \leq 6$  and consider the pool hypergraph of stage 1. Assume that some  $\geq 3$ -hyperedge  $e$  exists. If  $e$  is positive,  $e$  explains 3 or more positive pools. Since  $t(6, 1, 2) = 4$ , the searcher needs 4 more tests for the 2nd defective. Hence the pool hypergraph is merely a graph. Next, let  $p$  be a pool vertex with degree 1. Let  $p$  be negative and apply Lemma 3. The edge incident to  $p$  is negative, hence still 2 defectives among 6 elements must be found, and 1 pool is used up. Thus  $t(6, 2, 2) + 1 = 6 + 1 = 7$  tests are needed. Hence the minimum degree is 2. Assume that parallel edges exist, that is, 2 pools share 2 or more elements. Declare these 2 pools positive and apply Lemma 3 together with 1. Since still at least  $11 > 2^3$  candidate 2-sets remain, and 2 pools are used up, the searcher needs  $2 + 3 + 2$  pools. Altogether, the pool hypergraph must be a graph of minimum degree 2 without parallel edges. It has at most 5 pool vertices, since 6 pools would forbid a 2nd stage and require  $t(7, 2, 1) = 6$ , contradicting the known  $t(7, 2, 1) = 7$  [15].

Next we can show that cycles  $C_3, C_4, C_5$  together with edges or loops for the other elements always create bad induced subgraphs that enforce too many tests in stage 2 due to our Lemmas. Therefore the pool graph is a forest, perhaps with loops, and all leaves must have loops due to the minimum degree 2. Again, leaves at any distance create bad induced subgraphs, thus no edges other than loops can exist. (Details are omitted due to lack of space.)

It follows that all pool vertices are isolated and have at least 2 loops each. Since 2 such vertices imply already 4 more tests, we can have at most 2 pool vertices and  $p_0$ . By the pigeonhole principle, 2 vertices have at least 2 and 3 loops, respectively (or even 1 vertex has 5 loops). Hence the candidate graph contains  $K_{2,3}$ . Using Lemma 5, at least  $t(3, 1, 1) + 2 = 5$  more tests are required. Thus we can have only one pool vertex  $p_1$ . Since at least 2 loops are at  $p_1$ , the candidate graph contains  $K_{2,5}$ , and  $t(5, 1, 1) + 2 = 6$  more tests are needed by Lemma 5.

**$t(7, 3, 1+) = 7$ .** Follows from the 35 candidate 3-sets by Lemma 1.

**t(8, 2, 2+) = 7.** First we show that 7 tests are needed even adaptively. If we begin with a 2-pool and it is negative, then  $t(6, 2, 6) = 6$  enforces 6 further tests. If we begin with a 3-pool and it is positive, the  $18 > 2^4$  candidate 2-sets and Lemma 1 enforce 6 more tests. Due to Lemma 2 we need not consider more cases. **(S)** To manage with 7 tests in 2 stages, we test only one 2-pool in stage 1. If negative, we test the other 6 elements individually in stage 2. If positive, we test the 2 elements in this pool individually, and simultaneously we find up to 1 defective among the other 6 elements using  $t(6, 1, 1) = 4$  tests. Note that this strategy also reports if  $|P| > 2$ .

**t(8, 2, 1) = 8** is implicit in [15].

**t(8, 3, 1+) = 8.** Consider the first test of an adaptive strategy. A negative 1-pool enforces 7 more tests since  $t(7, 3, 7) = 7$ . A positive 2-pool leaves us with  $36 > 2^5$  candidate 3-sets. Apply Lemma 1 and finally Lemma 2.

**t(9, 2, 2+) = 7.** Clearly we only have to show the upper bound for  $s = 2$ . **(S)** Let the pool graph in stage 1 be  $K_4$  (6 edges) plus 3 loops at  $p_0$ . It is impossible that exactly 1 pool responds positive. If all pools are negative, then so are the edges in the  $K_4$ . If exactly 2 pools are positive, then exactly the edge between them is positive. In the above cases there remain only 3 candidates (loops) in stage 2. If 3 or 4 pools are positive, then we get exactly 3 candidate 2-sets, using edges of the  $K_4$  vertices only. Thus Lemma 6 applies.

**t(9, 2, 1) = 9** is known from [15].

**t(9, 3, 1+) = 9.** We systematically check the tree of all adaptive strategies and give test answers + or – to the searcher’s disadvantage. For certain paths in the search tree we find that the searcher is forced to apply too many tests, using earlier bounds and Lemma 1. By Lemma 2 and exploring symmetric cases we can prune most of the tree. Details are omitted due to lack of space, however we remark that our proof has to check just 11 paths, compared to the host of possible strategies and answers.

**t(10, 2, 3+) = 7.** We have  $t(10, 2, 3) \geq t(8, 2, 3) = 7$ . **(S)** For the upper bound we use the following pool graph in stage 1. Take a  $K_4$ , but delete the edge  $p_1 p_4$  and insert a loop at  $p_1$  and  $p_4$  instead. These 7 elements are complemented with 3 loops at  $p_0$ . (Here we particularly emphasize that this pooling design is far from being obvious, we found it after excluding other options with the help of our lower-bound methods. The same remark applies to other cases as well.) As can be quickly checked one by one, all conceivable test outcomes yield one of the following cases (possibly with further dummy elements): at most 3 candidate sets; or 1 recognized defective and at most 4 candidates for a 2nd one; or the candidate graph  $K_{1,3} + e$ . Using  $t(4, 1, 2) = 3$ , Lemma 6, and Lemma 7 for the next 2 stages, we can solve all cases in 2 more stages with 3 more tests.

**t(10, 2, 2) = 8.** **(S)** For  $t(10, 2, 2) \leq 8$  use  $K_5$  as the pool graph in stage 1. It is easy to check that 3 more tests are always enough in stage 2.

Now assume that  $t(10, 2, 2) \leq 7$ . In the same way as for  $t(7, 2, 2)$  we can show, due to  $t(9, 1, 2) = 5$ ,  $t(7, 2, 2) = 7$ , and Lemma 1, that the pool hypergraph in

stage 1 is a graph without parallel edges, now with minimum degree 4. The latter implies that at most 5 pool vertices exist. Since a  $C_3$  with loop implies 4 more tests, no further pool vertices can exist. By minimum degree 4, each vertex has at least 2 loops. If all 3 pools are positive, the 9 candidate 2-sets yield 5 more tests by Lemma 1. A  $C_3$  without loop would mean that any vertex of the  $C_3$  has also 2 neighbors outside, leading to 5 pools. But the  $C_3$  requires already 3 more tests. Hence no  $C_3$  can exist. Lemma 5 gives that 2 vertices with 2 loops each require 4 more tests. Thus, in a  $C_4$  at least 3 vertices must be incident to further edges. To avoid  $C_3$ , at least 6 pool vertices are needed. Hence no  $C_4$  can exist either. A  $C_5$  cannot exist, since further edges create smaller cycles, and 2 loops per vertex are too many. Hence the pool graph is a forest, with at least 3 loops at every leaf or isolated vertex. Since 2 vertices with 3 and 2 loops imply 5 tests (Lemma 5), at most 2 pool vertices exist. If  $p_1$  and  $p_2$  exist, we choose 2 loops at  $p_1$  and 4 loops at  $p_2$  (or possibly the edge  $p_1p_2$  instead of 1 loop) to get a candidate graph  $K_{2,4}$  that needs  $t(4, 1, 1) + 2 = 6$  more tests. If only  $p_1$  exists, we choose 2 loops at  $p_1$  and the 8 other loops from  $p_1$  or  $p_0$  to get a candidate graph  $K_{2,8}$  that needs  $t(8, 1, 1) + 2 = 7$  more tests.

$t(10, 2, 1) = 9$  follows from [15].

$t(10, 3, 3+) = 9$ . The lower bound holds since  $t(9, 3, 9) = 9$ . **(S)** Our strategy tests 3 disjoint 2-pools in stage 1. If at most 1 pool is positive, the at most 6 candidate elements are tested individually. If 2 pools are positive, 2 tests recognize 2 defectives in stage 2 (as in the  $t(7, 2, 3+)$  strategy). To find a possible 3rd defective among the other 6 elements in stage 3 we use  $t(6, 1, 1) = 4$ . If all 3 pools are positive, we determine the 3 defectives by 3 tests in stage 2, and 1 final test is used to discard the negative elements or report  $|P| > 3$ .

$t(10, 4, 1+) = 10$ . Consider the first test of an adaptive strategy. A negative 1-pool enforces 9 more tests since  $t(9, 4, 9) = 9$ . In the case of a positive 2-pool, even revealing a defective means that 3 defectives out of 9 elements must be found, but  $t(9, 3, 9) = 9$ . By Lemma 2, this case distinction is complete.

## 6 The Case of Two Defectives and Two Stages

Our  $t(9, 2, 2)$  strategy readily extends to larger  $n$  as follows. Let  $m$  be the smallest integer with  $\binom{m}{2} + 3 \geq n$ . Using  $K_m$  plus 3 loops at  $p_0$  (or any subset of this edge set) as the pool graph in stage 1, the same reasoning as for  $t(9, 2, 2)$  yields  $t(n, 2, 2) \leq m + 3$ . Although this test number grows as  $\Theta(\sqrt{n})$ , it is optimal (or close to optimal) for surprisingly many  $n$ . Below we report some exact results and their lower-bound arguments; note that Lemma 3 is silently used.

$t(8, 2, 2) = t(13, 2, 2) = 8$  was already shown.

$t(15, 2, 2) = t(18, 2, 2) = 9$ . Assume that  $t(15, 2, 2) \leq 8$ , and consider the pools in stage 1. A positive  $\geq 7$ -pool allows  $70 > 2^6$  candidate 2-sets, hence by Lemma 1 the remaining 7 tests would not be sufficient. A negative  $\leq 5$ -pool together with  $t(10, 2, 2) = 8$  yields 9 tests. Hence only 6-pools can be used. But 2 intersecting

positive 6-pools allow at least  $25 + 14 = 39 > 2^5$  candidate 2-sets, and 2 disjoint positive 6-pools allow  $36 > 2^5$  candidate 2-sets, such that the remaining 6 tests are not sufficient. Hence only one 6-pool may be used. If it responds positive, there remain  $69 > 2^6$  candidate 2-sets, implying 8 more tests.

**t(22, 2, 2) = t(24, 2, 2) = 10.** Assume that  $t(22, 2, 2) \leq 9$ , and consider the pools in stage 1. A positive  $\geq 8$ -pool allows  $28 + 8 \cdot 14 = 140 > 2^7$  candidate 2-sets, leading to 10 tests in total. A negative  $\leq 7$ -pool together with  $t(15, 2, 2) = 9$  yields 10 tests, too. Hence no pool size is usable.

Remarkably, the  $K_m$  plus 3 loops strategy misses the simple antichain lower bound of Lemma 1 by at most 1 test up to  $n = 31$ , and by at most 2 tests up to  $n = 58$ . However, clearly for large enough  $n$  some  $O(\log n)$  tests strategy takes over, and it is interesting to ask what constant factor we can achieve.

**Theorem 3.** *We have  $t(n, 2, 2) \leq 2.5 \log_2 n + o(\log_2 n)$ , and the trivial lower bound  $t(n, 2, 2) \geq 2 \log_2 n - 1$ .*

*Proof.* We encode the  $n$  elements as vectors over an alphabet of 4 symbols. The code length is  $m = \log_4 n = 0.5 \log_2 n$ . In stage 1 we test  $4 \cdot 0.5 \log_2 n = 2 \log_2 n$  pools, each consisting of all elements that share a fixed symbol at a fixed position. At most 2 of the 4 pools for every position can be positive, otherwise  $|P| > d$  is confirmed. Thus we have at most  $2^m$  candidate elements, and by construction they form disjoint candidate 2-sets. (In the case  $|P| = 1$  the only defective is already recognized.) We have  $2^m = 2^{0.5 \log_2 n} = \sqrt{n}$ . Thus, searching for the 2 defectives is equivalent to searching for 1 defective in a  $\leq \sqrt{n}/2$ -set. This requires  $t(\lceil \sqrt{n}/2 \rceil, 1, 1) = 0.5 \log_2 n + o(\log_2 n)$  tests in stage 2.  $\square$

## 7 Conclusions

We provided methods that make the construction of provably optimal multistage group testing strategies for specific input parameters manageable. The ultimate goal for further research would be a smooth transition from optimal strategies for small  $n$  to asymptotically optimal ones. The tools may be further refined to limit the case inspections even more. It would also be helpful to partly automatize the search and leave case inspections to computer programs. However this is not straightforward. To avoid combinatorial explosion we must generate certain set families up to symmetries. Among the open theoretical questions we mention the most intriguing ones: Is the nonadaptive test number additive for the product of candidate hypergraphs (see Lemma 5)? Does there exist, for every  $d$ , some  $s$  such that  $t(n, d, s) = t(n, d, n)$ ? In [4] we gave an affirmative answer only for  $d = 1$ , namely  $s = 2$ .

**Acknowledgments.** Support came from the Swedish Research Council (Vetenskapsrådet), grant 2010-4661, “Generalized and fast search strategies for parameterized problems”. An early version has been presented at the informal workshop “Search Methodologies III” 2012, organized by Christian Deppe and Ferdinando Cicalese at the Center for Interdisciplinary Research, University of Bielefeld.

## References

1. Balding, D.J., Torney, D.C.: Optimal Pooling Designs with Error Detection. *J. Comb. Theory A* 74, 131–140 (1996)
2. Chen, H.B., Hwang, F.K.: Exploring the Missing Link Among  $d$ -Separable,  $\bar{d}$ -Separable and  $d$ -Disjunct Matrices. *Discr. Appl. Math.* 155, 662–664 (2007)
3. Cheng, Y., Du, D.Z.: New Constructions of One- and Two-Stage Pooling Designs. *J. Comp. Biol.* 15, 195–205 (2008)
4. Damaschke, P.: Optimal Randomized Group Testing: A Canonical Form and the One-Defective Case. In: Cicalese, F., Porat, E. (eds.) ICALP2011GT (informal proceedings), Zürich, pp. 55–67 (2011)
5. De Bonis, A., Di Crescenzo, G.: Combinatorial Group Testing for Corruption Localizing Hashing. In: Fu, B., Du, D.-Z. (eds.) COCOON 2011. LNCS, vol. 6842, pp. 579–591. Springer, Heidelberg (2011)
6. De Bonis, A., Gasieniec, L., Vaccaro, U.: Optimal Two-Stage Algorithms for Group Testing Problems. *SIAM J. Comp.* 34, 1253–1270 (2005)
7. Du, D.Z., Hwang, F.K.: Combinatorial Group Testing and Its Applications. Series on Appl. Math., vol. 18. World Scientific (2000)
8. Du, D.Z., Hwang, F.K.: Pooling Designs and Nonadaptive Group Testing. Series on Appl. Math., vol. 18. World Scientific (2006)
9. Dyachkov, A.G., Rykov, V.V.: Bounds on the Length of Disjunctive Codes. *Problems of Info. Transmission* 18, 7–13 (1982) (in Russian)
10. Eppstein, D., Goodrich, M.T., Hirschberg, D.S.: Improved Combinatorial Group Testing Algorithms for Real-World Problem Sizes. *SIAM J. Comp.* 36, 1360–1375 (2007)
11. Fang, J., Jiang, Z.L., Yiu, S.M., Hui, L.C.K.: An Efficient Scheme for Hard Disk Integrity Check in Digital Forensics by Hashing with Combinatorial Group Testing. *Int. J. Digital Content Technol. and its Appl.* 5, 300–308 (2011)
12. Fischer, P., Klasner, N., Wegener, I.: On the Cut-off Point for Combinatorial Group Testing. *Discr. Appl. Math.* 91, 83–92 (1999)
13. Gao, H., Hwang, F.K., Thai, M.T., Wu, W., Znati, T.: Construction of  $d(H)$ -Disjunct Matrix for Group Testing in Hypergraphs. *J. Comb. Opt.* 12, 297–301 (2006)
14. Goodrich, M.T., Hirschberg, D.S.: Improved Adaptive Group Testing Algorithms with Applications to Multiple Access Channels and Dead Sensor Diagnosis. *J. Comb. Optim.* 15, 95–121 (2008)
15. Huang, S.H., Hwang, F.K.: When is Individual Testing Optimal for Nonadaptive Group Testing? *SIAM J. Discr. Math.* 14, 540–548 (2001)
16. Mézard, M., Toninelli, C.: Group Testing With Random Pools: Optimal Two-Stage Algorithms. *IEEE Trans. Info. Th.* 57, 1736–1745 (2011)
17. Spencer, J.: Minimal Completely Separating Systems. *J. Combin. Theory* 8, 446–447 (1970)
18. <http://www.redcrossblood.org/learn-about-blood/what-happens-donated-blood/blood-testing> (version as of January 2013)
19. Xuan, Y., Shin, I., Thai, M.T., Znati, T.: Detecting Application Denial-of-Service Attacks: A Group-Testing-Based Approach. *IEEE Trans. Par. Distr. Syst.* 21, 1203–1216 (2010)
20. Zhang, B.: Group Testing Regression Models. Dissertation, Dept. of Statistics, Univ. of Nebraska, Lincoln (2012)

# A Linear Edge Kernel for Two-Layer Crossing Minimization

Yasuaki Kobayashi, Hirokazu Maruta, Yusuke Nakae, and Hisao Tamaki

Meiji University, Kawasaki, Japan 214-8571  
`{yasu0207,maruta,nakae,tamaki}@cs.meiji.ac.jp`

**Abstract.** We consider a simple generalization of two-layer crossing minimization problem (TLCM) called leaf-edge-weighted TLCM (LEW-TLCM), where we allow positive weights on edges incident to leaves, and show that this problem admits a kernel with  $O(k)$  edges provided that the given graph is connected. As a straightforward consequence, LEW-TLCM (and hence TLCM) has a fixed parameter algorithm that runs in  $2^{O(k \log k)} + n^{O(1)}$  time which improves on the previously best known algorithm with running time  $2^{O(k^3)}n$ .

## 1 Introduction

A *two-layer drawing* of a bipartite graph  $G$  with bipartition  $(X, Y)$  of vertices places vertices in  $X$  on one line and those in  $Y$  on another line parallel to the first and draws edges as straight line segments between these two lines. We call these parallel lines *layers* of the drawing. A *crossing* in a two-layer drawing is a pair of edges that intersect each other at a point not representing a vertex. Note that the set of crossings in a two-layer drawing of  $G$  is completely determined by the order of the vertices in  $X$  on one layer and the order of the vertices in  $Y$  on the other layer. The problem to find a two-layer drawing whose crossing number is the minimum is called *two-layer crossing minimization*, TLCM for short. This problem is known to be NP-hard [12] (although it is polynomial time solvable for trees [14] and permutation graphs [15]). We consider this problem and its generalization on a parameterized perspective described as follows. An edge is a *leaf edge* if it is incident to a leaf (a vertex of degree one); a *non-leaf edge* otherwise.

Two-Layer Crossing Minimization (TLCM)

**Instance:** bipartite graph  $G = (V(G), E(G))$

**Parameter:**  $k$

**Question:** Is there a two-layer drawing of  $G$  with at most  $k$  crossings?

Leaf-Edge-Weighted Two-Layer Crossing Minimization (LEW-TLCM)

**Instance:** bipartite graph  $G = (V(G), E(G))$ , function  $w : E(G) \rightarrow \mathbb{N}$  with  $w(e) = 1$  for every non-leaf edge  $e \in E(G)$

**Parameter:**  $k$

**Question:** Is there a two-layer drawing of  $G$  with crossings of total weight at most  $k$ , where a crossing  $(e, e')$  has weight  $w(e)w(e')?$

Clearly, LEW-TLCM is a generalization of TLCM. In this paper, we give kernelizations for these problems.

A *kernelization* [4] for a parameterized problem is an algorithm that, given an instance  $I$  and a parameter  $k$ , computes an instance  $I'$  and a parameter  $k'$  of the same problem in time polynomial in  $k$  and the size of  $I$  such that

1.  $(I, k)$  is feasible if and only if  $(I', k')$  is feasible,
2. the size of  $I'$  is bounded by a computable function  $f$  in  $k$ , and
3.  $k'$  is bounded by a function in  $k$ .

When the function  $f$  is polynomial, we call the algorithm a *polynomial kernelization* and its output a *polynomial kernel*.

TLCM is a special case of a problem called  $h$ -layer crossing minimization which decides if a given graph has an  $h$ -layer drawing with at most  $k$  crossings. This problem is fixed parameter tractable [6] when parameterized by  $k + h$ . The running time of the algorithm of [6] is  $2^{O((h+k)^3)}n$ . Besides having a large exponent in the running time, this algorithm is rather complicated, involving, in particular, the fixed parameter algorithm for pathwidth due to Bodlaender and Kloks [1,2] and is not easy to implement. It is natural to ask if we can obtain a simpler and faster fixed parameter algorithm for the special case of  $h = 2$ , namely TLCM. To the best of the present authors' knowledge, neither a faster algorithm for TLCM than the one given in [6] nor its polynomial kernelization is previously known.

In contrast to TLCM, several fixed parameter algorithms are known for *two-layer planarization* (TLP), where the objective is to find a subset of edges of size at most  $k$  of the given graph whose removal enables a two-layer drawing without any crossings. This problem is fixed parameter tractable [5] and can be solved in time  $O(k \cdot 3.562^k + n)$  where  $n$  is the number of vertices of the given graph [16]. Moreover, a kernel with  $O(k)$  edges for TLP is known [5].

Another related problem is *one-sided crossing minimization* (OSCM), which asks for a two-layer drawing of the given bipartite graph with minimum number of crossings, but with the vertex order in one layer fixed as part of the input. OSCM is also NP-hard [10]. The fixed parameter tractability of OSCM seems to be better-studied [8,7,11,13] than TLCM. More specifically, [8] gives the first fixed parameter algorithm, [7] gives a faster fixed parameter algorithm and a kernel with  $O(k^2)$  edges, and [11] and [13] give subexponential fixed parameter algorithms with running time  $2^{O(\sqrt{k} \log k)} + n^{O(1)}$  and  $O(3^{\sqrt{2k}} + n)$ , respectively.

Our results are as follows.

**Theorem 1.** *TLCM admits a kernel with  $O(k^2)$  edges provided that the given graph is connected.*

**Theorem 2.** *LEW-TLCM admits a kernel with  $O(k)$  edges provided that the given graph is connected.*

The second theorem implies a fixed parameter algorithm with running time  $2^{O(k \log k)} + n^{O(1)}$  for both TLCM and LEW-TLCM via the standard approach:

given an instance of TLCM or LEW-TLCM, we construct a kernel with  $O(k)$  edges in LEW-TLCM for each connected component and then do an exhaustive search to find a solution for each of these kernels. Note that all but  $k$  connected components are crossing-free, assuming that the given instance is feasible, which can be detected by a simple linear time algorithm [9].

We remark that, although some of the lemmas needed for the kernelization are non-trivial, the kernelization algorithm itself is quite simple and easy to implement.

The rest of this paper is organized as follows. In section 2, we give preliminaries for TLCM. In section 3, we describe a kernelization for TLCM whose output has  $O(k^2)$  edges, proving some lemmas necessary for this kernelization. In section 4, we show that the same method works for LEW-TLCM and gives a kernelization whose output has  $O(k)$  edges. Finally, Section 5 contains the conclusion.

## 2 Preliminaries

Let  $G$  be a bipartite graph with a prescribed bipartition of the vertex set. We denote by  $V(G)$  the set of vertices of  $G$ , by  $(X(G), Y(G))$  the bipartition of  $V(G)$ , and by  $E(G) \subseteq X(G) \times Y(G)$  the set of edges of  $G$ . We also view  $G$  as a triple  $(X(G), Y(G), E(G))$ . For each vertex  $v \in V(G)$ , we denote by  $d(v)$  the degree of  $v$ . A *leaf* is a vertex  $v$  with  $d(v) = 1$ . We call an edge a *leaf edge* if the edge is incident to a leaf; otherwise a *non-leaf edge*. For an edge  $e$  in  $G$ , the graph  $G - e$  is a subgraph obtained from  $G$  by deleting  $e$ . A *cut vertex* (a *bridge*) is a vertex (an edge) whose removal increases the number of components. A *block* is a maximal connected subgraph without a cut vertex. We say that a block is *trivial* if it has at most two vertices. Otherwise, the block is *non-trivial*. For each subset  $U$  of  $V(G)$ , we denote by  $G[U]$  the subgraph of  $G$  induced by  $U$ . For each subgraph  $G'$  of  $G$ , we denote  $X(G) \cap V(G')$  by  $X(G')$  and  $Y(G) \cap V(G')$  by  $Y(G')$ .

A two-layer drawing  $\mathcal{D}$  of  $G$  is defined as a triple  $(G, <_X, <_Y)$  where  $<_X$  and  $<_Y$  are total orders on  $X(G)$  and  $Y(G)$ , respectively. The number of crossings in  $\mathcal{D}$  is the number of pairs of edges that intersect each other:

$$bcr(\mathcal{D}) = \sum_{\{u,v\} \in E(G)} |\{(x,y) \in E(G) : x <_X u, v <_Y y\}|.$$

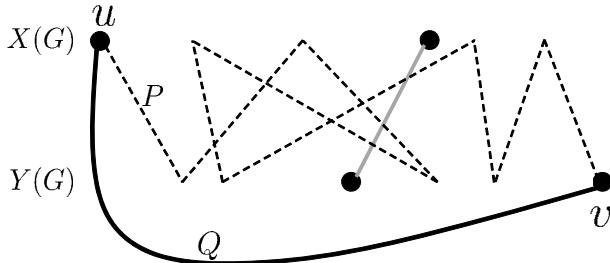
The *bipartite crossing number* of  $G$ , denoted by  $bcr(G)$ , is the minimum number of crossings over all two-layer drawings of  $G$ . For each subgraph  $G'$  of  $G$ , we denote by  $\mathcal{D} | G'$  the two-layer drawing of  $G'$  in which the vertices of  $V(G')$  are placed in the same order as in  $\mathcal{D}$ . For distinct vertices  $u, v \in X(G)$ , we say that a vertex  $u$  is *to the left* (*right*) of  $v$  in  $\mathcal{D}$  if  $u <_X v$  ( $v <_X u$ ). A vertex  $u \in X(G)$  is the *leftmost* (*rightmost*) in  $\mathcal{D}$  if  $u <_X v$  ( $v <_X u$ ) for all  $v \in X(G) \setminus \{u\}$ . We may omit the reference to  $\mathcal{D}$  when it is clear from the context. We use similar terminology for vertices in  $Y(G)$ .

### 3 A Kernel with $O(k^2)$ Edges for TLCM

In this section, we give a kernelization for TLCM whose output has  $O(k^2)$  edges. The same approach is extended to a kernelization for LEW-TLCM in the next section. This kernelization is based on some lower bound on the bipartite crossing number and some reduction rule on bridges. First, we show a technical lemma.

**Lemma 1.** *Let  $\mathcal{D}$  be a two-layer drawing of a bipartite graph  $G$  and let  $P$  be a path in  $\mathcal{D}$  from a leftmost vertex  $u$ , either in  $X(G)$  or  $Y(G)$ , to a rightmost vertex  $v$ , either in  $X(G)$  or  $Y(G)$ . Then, each edge not incident to  $V(P)$  has a crossing with some edge in  $P$ .*

*Proof.* Consider an arbitrary geometric representation of  $\mathcal{D}$  and connect  $u$  and  $v$  with a curve  $Q$  not intersecting any edge in the drawing. Then, the closed curve consisting of  $Q$  and the polygonal curve representing  $P$  divides the plane into several regions. One of them contains  $X(G) \setminus V(P)$  in its interior and another region, distinct from the first, contains  $Y(G) \setminus V(P)$  in its interior. Therefore, each edge not incident to  $V(P)$  must intersect with the closed curve and hence cross some edge of  $P$ , since it does not intersect with  $Q$ . See Fig. 1 for an example.  $\square$



**Fig. 1.** The dotted polygonal curve indicates a path  $P$  and the solid black line indicates a curve  $Q$ . Each edge not incident to  $V(P)$  has a crossing with  $P$ .

**Lemma 2.** *Let  $G$  be a biconnected bipartite graph. Then  $bcr(G) \geq \frac{|E(G)|-1}{3}$ .*

*Proof.* We prove the statement by induction on the number of edges. The base case where  $|E(G)| = 4$  holds since the edge minimum biconnected bipartite graph is a cycle of length four, which must have a crossing.

Suppose  $|E(G)| \geq 5$ . Let  $\mathcal{D}$  be a two-layer drawing of  $G$  with the minimum number of crossings. We consider the three cases: (1)  $G$  is triconnected, (2)  $G$  is a cycle, or (3)  $G$  is biconnected but neither is triconnected nor is a cycle.

(1) Let  $e$  be an edge of  $G$  such that  $e$  has a crossing with another edge of  $G$ . Since  $G$  is triconnected,  $G - e$  is biconnected. Then we have, by the induction hypothesis,  $bcr(\mathcal{D}) \geq bcr(\mathcal{D} \setminus (G - e)) + 1 \geq \frac{|E(G-e)|-1}{3} + 1 > \frac{|E(G)|-1}{3}$ .

(2) Let  $u$  be the leftmost vertex in  $X(G)$ . If  $|E(G)|/2$  is even (odd), then let  $v$  be the rightmost vertex in  $Y(G)$  ( $X(G)$ ). Then the two paths of  $G$  between  $u$

and  $v$  cannot have the same length because of the parity. Let  $P$  be the shorter of these two paths. Then, we have  $|E(P)| \leq \frac{|E(G)|}{2} - 1$ . By Lemma 1, each of the at least  $\frac{|E(G)|}{2} - 1$  edges not incident to  $V(P)$  has a crossing with some edge of  $P$ , which is at least  $\frac{|E(G)|-1}{3}$  when  $|E(G)| \geq 5$ .

(3) In this case, there are two vertices  $u, v$  where  $G[V(G) \setminus \{u, v\}]$  is disconnected. Let  $V_1, V_2, \dots, V_d$  be the components of  $G[V(G) \setminus \{u, v\}]$  and  $G_i = G[V_i \cup \{u, v\}]$  for  $1 \leq i \leq d$ . We assume that at least one of  $G_i$  is not a path: we may choose  $u$  and  $v$  so that this assumption is satisfied since  $G$  is not a cycle.

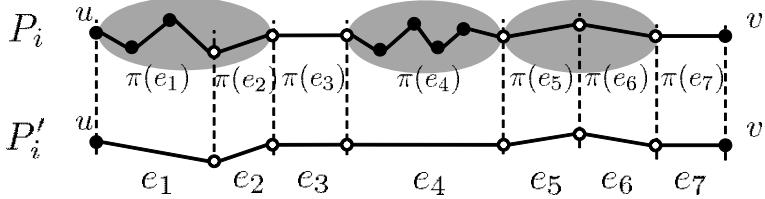
Suppose first that  $G$  has an edge between  $u$  and  $v$ . Then, for each  $i$ ,  $G_i$  is biconnected and hence, using the induction hypothesis, we have  $bcr(G_i) \geq \frac{1}{3}(|E(G_i)| - 1)$ . Therefore, we have

$$\begin{aligned} bcr(\mathcal{D}) &\geq \sum_{1 \leq i \leq d} bcr(\mathcal{D} \mid G_i) \\ &\geq \sum_{1 \leq i \leq d} bcr(G_i) \\ &\geq \frac{1}{3} \sum_{1 \leq i \leq d} (|E(G_i)| - 1) \\ &= \frac{1}{3} (|E(G)| + d - 1 - d) \\ &= \frac{1}{3} (|E(G)| - 1) \end{aligned}$$

and we are done.

Suppose next that  $G$  does not have an edge between  $u$  and  $v$ . For  $1 \leq i \leq d$ , we denote by  $\mathcal{B}_i$  the set of blocks of  $G_i$  and let  $\mathcal{B} = \bigcup_{1 \leq i \leq d} \mathcal{B}_i$ . For each  $i$ , since  $G_i$  becomes biconnected when we add an edge between  $u$  and  $v$ ,  $G_i$  contains a path  $P_i$  between  $u$  and  $v$  that goes through all the blocks of  $G_i$ . Let  $W_i$  be a minimal subset of  $V(P_i) \setminus \{u, v\}$  that contains all the cut vertices of  $G_i$  and partitions  $P_i$  into paths of odd length. See Fig. 2 for an example. In  $P_i$ , replace each minimal subpath between vertices in  $W_i \cup \{u, v\}$  by a single edge and let  $P'_i$  be the resulting path on  $W_i \cup \{u, v\}$ . For each edge  $e$  of  $P'_i$ , we let  $\pi(e)$  denote the subpath of  $P_i$  that  $e$  replaces.

Let  $H$  be the graph that is the union of  $P'_i$ ,  $1 \leq i \leq d$ . Observe that  $V(H) \subseteq V(G)$  and  $H$  is bipartite with bipartition  $(X(G) \cap V(H), Y(G) \cap V(H))$ . Let  $Q$  be the set of crossings  $(e, e')$  in  $\mathcal{D}$  such that  $e$  and  $e'$  belong to distinct blocks in  $\mathcal{B}$ . We claim that  $|Q| \geq bcr(H)$ . To see this, consider the two-layer drawing  $\mathcal{D}'$  of  $H$  induced by  $\mathcal{D}$ : the vertices of  $H$  are ordered in  $\mathcal{D}'$  as they are ordered in  $\mathcal{D}$ . Suppose edges  $e$  and  $e'$  of  $H$  cross each other in  $\mathcal{D}'$ . Then, considering the relative positions of the end-vertices of  $e$  and  $e'$  which are exactly the end-vertices of the paths  $\pi(e)$  and  $\pi(e)'$ , we see that  $\pi(e)$  and  $\pi(e)'$  must intersect in  $\mathcal{D}$ . Since  $\pi(e)$  and  $\pi(e)'$  are vertex disjoint, this intersection must be a crossing of edges. Moreover,  $\pi(e)$  and  $\pi(e)'$  belong to two different blocks in  $\mathcal{B}$  (otherwise  $e$  and  $e'$  are adjacent and do not cross). Therefore, each crossing of  $\mathcal{D}'$  gives rise to a distinct crossing in  $Q$  and hence we have  $|Q| \geq bcr(\mathcal{D}') \geq bcr(H)$  as



**Fig. 2.** An example of paths \$P\_i\$ and \$P'\_i\$. Shaded ovals indicate non-trivial blocks and white circles indicate the vertices in \$W\_i\$. The length of a path \$\pi(e)\$ for \$e \in E(P'\_i)\$ is odd.

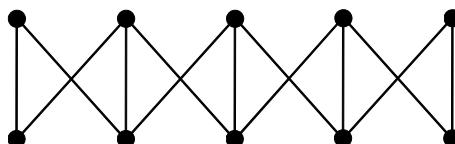
claimed. Since \$H\$ is biconnected and \$|E(H)| < |E(G)|\$ by the assumption that at least one of \$G\_i\$ is not a path, we have \$bcr(H) \geq \frac{|E(H)|-1}{3}\$ by the induction hypothesis. It follows that \$bcr(H) \geq \frac{|\mathcal{B}|-1}{3}\$ since \$|E(H)| \geq |\mathcal{B}|\$.

Now, let us turn to show the inequality in the statement of this lemma. Since \$|E(B)| < |E(G)|\$, for each \$B \in \mathcal{B}\$, we have \$bcr(B) \geq \frac{|E(B)|-1}{3}\$ by the induction hypothesis. Note that this inequality also holds when \$B\$ is a trivial block where \$bcr(B) = 0\$ and \$|E(B)| \leq 1\$. Consequently,

$$\begin{aligned} bcr(G) &\geq \sum_{B \in \mathcal{B}} bcr(B) + |Q| \\ &\geq \sum_{B \in \mathcal{B}} bcr(B) + bcr(H) \\ &\geq \sum_{B \in \mathcal{B}} \frac{|E(B)|-1}{3} + \frac{|\mathcal{B}|-1}{3} \\ &= \frac{|E(G)|-|\mathcal{B}|}{3} + \frac{|\mathcal{B}|-1}{3} \\ &= \frac{|E(G)|-1}{3}. \end{aligned}$$

□

The bound in this lemma is tight. See Fig. 3 for an example, which can be generalized to a biconnected graph with \$3k+1\$ edges and \$k\$ crossings for arbitrary \$k \geq 1\$.



**Fig. 3.** A biconnected graph \$G\$ with \$bcr(G) = \frac{|E(G)|-1}{3}

Fix a bipartite graph  $G$  and a positive integer  $k$ . We assume that  $G$  is connected since our goal is to establish Theorem 1. In the rest of this section, we show that  $G$  can be reduced to a graph  $G'$  such that  $bcr(G) = bcr(G')$  and  $|E(G')| \leq f(k)$ , where  $f(k) = O(k^2)$ , if  $bcr(G) \leq k$ . This implies a kernelization with  $O(k^2)$  edges: we output a trivial infeasible instance if  $|E(G')| > f(k)$ .

We say a bridge  $e$  of  $G$  is *order-inducing* if each of the two connected components of  $G - e$  has more than  $k$  edges; otherwise  $e$  is *non-order-inducing*. Let us note that every leaf edge is non-order-inducing. The following lemma justifies the name of order-inducing bridges.

**Lemma 3.** *Let  $e = (u, v)$  be an order-inducing bridge of  $G$  and let  $G_1$  and  $G_2$  be the connected components of  $G - e$  with  $u \in X(G_1)$  and  $v \in Y(G_2)$ . If  $bcr(G) \leq k$  then there is a two-layer drawing  $\mathcal{D} = (G, <_X, <_Y)$  with  $bcr(\mathcal{D}) \leq k$  in which the vertices of  $G_1$  are placed entirely to the left of the vertices of  $G_2$ . That is,  $a <_X b$  for  $a \in X(G_1), b \in X(G_2)$  and  $a <_Y b$  for  $a \in Y(G_1), b \in Y(G_2)$ .*

*Proof.* Let  $\mathcal{D}'$  be an arbitrary two-layer drawing with  $bcr(\mathcal{D}') \leq k$  in which  $G_1$  contains the leftmost vertex  $y_l$  of  $Y(G)$ .

First, we claim that the rightmost vertex  $x_r$  of  $X(G)$  in  $\mathcal{D}'$  is contained in  $X(G_2)$ . Assume otherwise, that  $G_1$  contains both  $y_l$  and  $x_r$ . Then, by Lemma 1, the path from  $y_l$  to  $x_r$  in  $G_1$  have a crossing with each edge in  $E(G_2)$ , contradicting the assumption that  $bcr(\mathcal{D}') \leq k$ .

We construct a two-layer drawing  $\mathcal{D} = (G, <_X, <_Y)$  as follows:

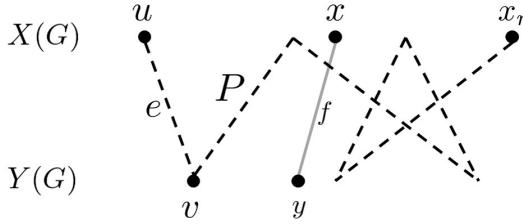
1.  $a <_X b$  for  $a \in X(G_1), b \in X(G_2)$ ,
2.  $a <_Y b$  for  $a \in Y(G_1), b \in Y(G_2)$ ,
3.  $\mathcal{D}' \mid G_1 = \mathcal{D} \mid G_1$ , and
4.  $\mathcal{D}' \mid G_2 = \mathcal{D} \mid G_2$ .

Clearly,  $<_X$  and  $<_Y$  are total orders on  $X$  and  $Y$  respectively. In the following we show that  $bcr(\mathcal{D}') \geq bcr(\mathcal{D})$ . Since each edge of  $G_1$  has no crossings with any edge of  $G_2$  in  $\mathcal{D}$  and the crossings within  $G_1$  and within  $G_2$  are preserved, to prove the inequality, it suffices to show that each edge  $f \in E(G_1)$  that crosses  $e$  in  $\mathcal{D}$  has at least one crossing with  $E(G_2) \cup \{e\}$  in  $\mathcal{D}'$ , together with the symmetric property for edges in  $G_2$ .

Let  $f = (x, y)$  be an edge of  $G_1$  that crosses  $e$  in  $\mathcal{D}$ . Let  $P$  be a path consisting of  $e$  and a path from  $v$  to  $x_r$  in  $G_2$ . Since  $e = (u, v)$  and  $f = (x, y)$  cross each other and  $y \in Y(G_1)$  is to the left of  $v \in Y(G_2)$  in  $\mathcal{D}$ ,  $x$  is to the right of  $u$  in  $\mathcal{D}$ . This order is the same in  $\mathcal{D}'$  as  $u, x \in X(G_1)$ . Moreover,  $x \neq x_r$  since  $x \in X(G_1)$  and  $x_r \in X(G_2)$ . Therefore,  $f = (x, y)$  crosses some edge of  $P$ , by an argument similar to the proof of Lemma 1. We are done since  $E(P) \subseteq E(G_2) \cup \{e\}$ . See Fig. 4 for an example.  $\square$

We say that an order-inducing bridge  $e$  of  $G$  is *contractable* if each end of  $e$  is incident to an order-inducing bridge distinct from  $e$  and is not incident to any non-leaf edge other than  $e$  and this order-inducing bridge.

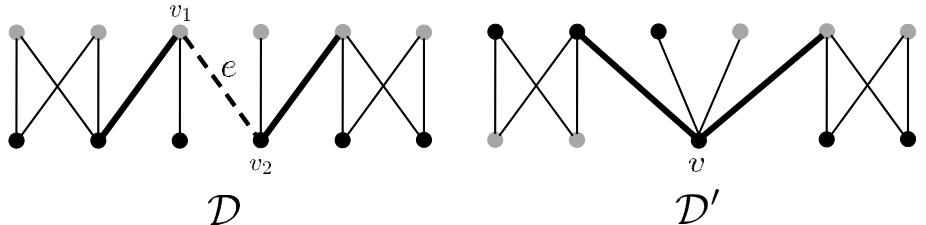
**Lemma 4.** *Suppose  $G$  has a contractable bridge  $e$  and let  $H$  be the result of contracting  $e$  in  $G$ . Then,  $bcr(G) \leq k$  if and only if  $bcr(H) \leq k$ .*



**Fig. 4.** Path  $P$  and edge  $f = (x, y)$  in the drawing  $\mathcal{D}'$ .  $P$ , which include  $e$ , is shown in dotted lines. Vertex  $x$  is to the right of  $u$  as it is in  $\mathcal{D}$ . Edge  $f$  crosses  $P$  no matter whether  $y$  is to the left or to the right of  $v$ .

*Proof.* Let  $e = (v_1, v_2)$  and let  $e_i$ ,  $i = 1, 2$ , be the order-inducing edge that is incident to  $v_i$  and is distinct from  $e$ . For  $i = 1, 2$ , let  $G_i$  be the connected component of  $G - e_i$  that does not contain  $e$ .

Suppose first that  $bcr(G) \leq k$  and let  $\mathcal{D}$  be a drawing of  $G$  with  $bcr(\mathcal{D}) = bcr(G)$ . Since  $e$ ,  $e_1$ , and  $e_2$  are order-inducing, by Lemma 3, we may assume that, in  $\mathcal{D}$ , the vertices of  $V(G_1)$  lie entirely to the left of  $v_1$  and  $v_2$  while the vertices of  $V(G_2)$  lie entirely to the right of  $v_1$  and  $v_2$ . Therefore, the edges of  $G_i$  have no crossings with edges not in  $E(G_i) \cup \{e_i\}$ , for  $i = 1, 2$ . Let  $\mathcal{D}'$  be the drawing of  $H$  naturally derived from  $\mathcal{D}$  as follows. Starting from the drawing  $\mathcal{D}$ , take its subdrawing  $\mathcal{D} | G'_1$ , where  $G'_1 = G[V(G_1) \cup \{v_1\}]$ , and flip it upside down, that is, place the vertices in  $X(G)$  on the layer for  $Y(G)$  and vice versa and keep the order of vertices within  $G'_1$ . Then, contract  $v_1$  and  $v_2$ , now in the same layer, into one vertex. Finally, place all the leaves adjacent to this contracted vertex between the drawings of  $G_1$  and  $G_2$ . Clearly, we have  $bcr(\mathcal{D}') = bcr(\mathcal{D})$ . See Fig. 5.



**Fig. 5.** An example of two-layer drawings  $\mathcal{D}$  and  $\mathcal{D}'$ . The dotted line indicates a contractable bridge and thick lines indicate order-inducing bridges.

To show the reverse direction, suppose  $bcr(H) \leq k$  and let  $\mathcal{D}'$  be a drawing of  $H$  with  $bcr(\mathcal{D}') = bcr(H)$ . Let  $v$  be the vertex of  $H$  into which  $e$  is contracted and  $e'_i$  the edge  $e_i$  with  $v_i$  replaced by  $v$ , for  $i = 1, 2$ . Since  $e_1$  and  $e_2$  are order-inducing in  $G$ ,  $e'_1$  and  $e'_2$  are order-inducing in  $H$ . Therefore, we may assume that, in  $\mathcal{D}'$ , the vertices of  $V(G_1)$  lie entirely to the left of  $v$  and  $V(G_2)$  while the vertices of  $V(G_2)$  lie entirely to the right of  $v$  and  $V(G_1)$ . By a conversion

that is an inverse of the above conversion from  $\mathcal{D}$  to  $\mathcal{D}'$ , we obtain a drawing  $\mathcal{D}$  of  $G$  such that  $bcr(\mathcal{D}) = bcr(\mathcal{D}')$ .  $\square$

Repeating the contraction of contractable bridges until there is no contractable bridges, we obtain a kernel of the given instance. The following lemma bounds the size of the kernel.

**Lemma 5.** *Suppose  $bcr(G) \leq k$  and  $G$  does not have any contractable bridge. Then, the number of non-leaf edges of  $G$  is at most  $10k + 3$ .*

This lemma follows from the following lemmas.

**Lemma 6.** *If  $bcr(G) \leq k$  then the number of edges belonging to non-trivial blocks is at most  $3k + 1$ .*

*Proof.* Let  $\mathcal{B}$  be the set of non-trivial blocks of  $G$ . If  $G$  has two edge disjoint subgraphs  $G_1$  and  $G_2$ , we have  $bcr(G) \geq bcr(G_1) + bcr(G_2)$ . Therefore, by Lemma 2,  $\sum_{B \in \mathcal{B}} |E(B)| \leq 3k + 1$ .  $\square$

**Lemma 7.** *If  $bcr(G) \leq k$  then the number of non-order-inducing bridges that are non-leaf edges is at most  $3k$ .*

*Proof.* Let  $\mathcal{D}$  be a two-layer drawing of  $G$  with the minimum number of crossings. Let  $P$  be a path of  $G$  between a leftmost vertex and a rightmost vertex in  $\mathcal{D}$ . There can be at most  $k$  non-leaf bridges that are not contained in  $P$ , since to each such bridge we may assign a distinct edge that is not incident to  $V(P)$ , which must have a crossing with some edge of  $P$ , by Lemma 1. Of all the non-leaf bridges contained in  $P$ , all but  $k$  from the left and  $k$  from the right are order-inducing, since each of them has at least  $k + 1$  edges ( $k$  non-leaf bridges and one additional edge) on both sides of it.  $\square$

**Lemma 8.** *If  $bcr(G) \leq k$  then the number of order-inducing bridges of  $G$  that are not contractable is at most  $4k + 2$ .*

*Proof.* Let  $\mathcal{D}$  be a two-layer drawing of  $G$  with the minimum number of crossings and let  $P$  be a path of  $G$  between a leftmost vertex and a rightmost vertex in  $\mathcal{D}$ . Note that every order-inducing bridge is contained in  $P$ : if there were an order-inducing bridge not contained in  $P$ , then all of the more than  $k$  edges in one side of that bridge would cross  $P$ .

Let  $e$  be an order-inducing bridge that is not contractable. Then  $e$  satisfies at least one of the following conditions: (1) at least one end of  $e$  is incident to some non-order-inducing bridge of  $G$ , (2) at least one end of  $e$  is incident to a non-trivial block of  $G$ , or (3) at least one end of  $e$  is incident to a non-leaf edge not contained in  $P$ .

There are at most two order-inducing bridges that are not contractable because of the reason (1). Since there are at most  $k$  non-trivial blocks, at most  $2k$  order-inducing bridges that are not contractable because of the reason (2). Since each non-leaf vertex not on  $P$  implies an edge not incident to  $V(P)$  which

has a crossing with some edge of  $P$ , there are at most  $k$  non-leaf edges incident to but not contained in  $P$ . Therefore, at most  $2k$  order-inducing bridges are not contractable because of the reason (3).

Summing up, the number of order-inducing bridges that are not contractable is at most  $4k + 2$ .  $\square$

Overall, we have Lemma 5.

To obtain a kernel with  $O(k^2)$  edges, we need an upper bound on the number of leaf edges of the reduced graph  $G$ . Note that we can assume each vertex is incident to at most  $k + 1$  leaf edges. This follows from the fact that there is a two-layer drawing  $\mathcal{D}$  of  $G$  with  $bcr(G) = bcr(\mathcal{D})$  such that the leaves of  $G$  with a common neighbor appear consecutively in  $\mathcal{D}$ . This means that, if a vertex has more than  $k$  leaf neighbors, then all but  $k + 1$  of them can be discarded without changing the feasibility of the instance. Therefore, Lemma 4 implies that the kernel obtained from a feasible instance has at most  $10k + 3 + (10k + 4)(k + 1) = 10k^2 + 24k + 7$  edges.

## 4 A Kernel with $O(k)$ Edges for LEW-TLCM

It is clear that the kernel for TLCM described in the previous section can be represented by an instance of LEW-TLCM with  $O(k)$  edges. Although this observation is sufficient for algorithmic purposes, we would like to say that LEW-TLCM has a kernel with  $O(k)$  edges for an aesthetic reason. To this end, we confirm that the lemmas in the previous section are applicable to instances of LEW-TLCM.

For each leaf-edge-weighted graph  $G$  with weight function  $w$ , let  $\text{unfold}(G, w)$  denote the unweighted graph equivalent to  $G$  with weight  $w$ : each vertex  $v$  that is incident to a leaf edge  $e$  in  $G$  is incident to  $w(e)$  leaf edges in  $\text{unfold}(G, w)$ .

To adapt the lemmas of the previous section to leaf-edge-weighted instances, we read “the number of edges” as “the sum of weights of edges” in the definitions and lemmas. Then the statements of the lemmas for a weighted instance  $(G, w, k)$  are equivalent to the statements for the unweighted instance  $(\text{unfold}(G, w), k)$  and hence do hold. Under this interpretation, the kernelization in the previous section works for an LEW-TLCM instance and produces a kernel with at most  $10k + 3 + (10k + 4) = 20k + 7$  edges.

## 5 Concluding Remarks

We have given an  $O(k)$  edge kernel for connected instances of a simple generalization of TLCM. Its consequences are not limited to the fixed parameter algorithm mentioned in the introduction, which applies a brute-force search to the kernel. There are other methods for exactly solving TLCM such as integer programming [17] and semidefinite programming [3]. Our kernelization is expected to broaden the class of instances practically solvable by such methods. We are planning an extensive experiments along this line.

## References

1. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing* 25(6), 1305–1317 (1996)
2. Bodlaender, H.L., Kloks, T.: Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms* 21(2), 358–402 (1996)
3. Chimani, M., Hungerländer, P., Jünger, M., Mutzel, P.: An SDP approach to multi-level crossing minimization. *Journal of Experimental Algorithmics* 3(3), 1–16 (2012)
4. Downey, R.G., Fellows, M.R.: Parameterized complexity. Springer (1999)
5. Dujmović, V., Fellows, M.R., Hallet, M., Kitching, M., Liotta, G., McCartin, C., Nishimura, N., Ragde, P., Rosamond, F., Whitesides, S., Wood, D.R.: A fixed parameter approach to two-layer planarization. *Algorithmica* 45(2), 159–182 (2006)
6. Dujmović, V., Fellows, M.R., Kitching, M., Liotta, G., McCartin, C., Nishimura, N., Ragde, P., Rosamond, F., Suderman, M., Whitesides, S., Wood, D.R.: On the parameterized complexity of layered graph drawing. *Algorithmica* 52(2), 267–292 (2008)
7. Dujmović, V., Fernau, H., Kaufmann, M.: Fixed parameter algorithms for one-sided crossing minimization revisited. *Journal of Discrete Algorithms* 6(2), 313–323 (2008)
8. Dujmović, V., Whitesides, S.: An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. *Algorithmica* 40(1), 15–31 (2004)
9. Eades, P., McKay, B.D., Wormald, N.C.: On an edge crossing problem. In: Proceedings of 9th Australian Computer Science Conference, pp. 327–334 (1986)
10. Eades, P., Wormald, N.C.: Edge crossings in drawings of bipartite graphs. *Algorithmica* 11(4), 379–403 (1994)
11. Fernau, H., Fomin, F.V., Lokshtanov, D., Mnich, M., Philip, G., Saurabh, S.: Ranking and drawing in subexponential time. In: Iliopoulos, C.S., Smyth, W.F. (eds.) IWOCA 2010. LNCS, vol. 6460, pp. 337–348. Springer, Heidelberg (2011)
12. Garey, M.R., Johnson, D.S.: Crossing number is NP-complete. *SIAM J. on Algebraic and Discrete Methods* 4(3), 312–316 (1982)
13. Kobayashi, Y., Tamaki, H.: A fast and simple subexponential fixed parameter algorithm for one-sided crossing minimization. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 683–694. Springer, Heidelberg (2012)
14. Shahrokhi, F., Sýkora, O., Székely, L.A., Vrt'o, I.: On bipartite Drawings and the linear arrangement problem. *SIAM Journal on Computing* 30(6), 1773–1789 (2001)
15. Spinrad, J., Brandstädt, A., Stewart, L.: Bipartite permutation graphs. *Discrete Applied Mathematics* 18(3), 279–292 (1987)
16. Suderman, M.: Layered graph drawing. PhD thesis, School of Computer Science, McGill University Montréal (2005)
17. Zheng, L., Buchheim, C.: A new exact algorithm for the two-sided crossing minimization problem. In: Dress, A.W.M., Xu, Y., Zhu, B. (eds.) COCOA 2007. LNCS, vol. 4616, pp. 301–310. Springer, Heidelberg (2007)

# A Linear-Time Algorithm for Computing the Prime Decomposition of a Directed Graph with Regard to the Cartesian Product<sup>\*</sup>

Christophe Crespelle<sup>1</sup>, Eric Thierry<sup>2</sup>, and Thomas Lambert<sup>3</sup>

<sup>1</sup> Université Claude Bernard Lyon 1, DANTE/INRIA, LIP UMR CNRS 5668,  
ENS de Lyon, Université de Lyon  
`christophe.crespelle@inria.fr`

<sup>2</sup> ENS de Lyon, LIP UMR CNRS 5668, Université de Lyon  
`{eric.thierry,thomas.lambert}@ens-lyon.fr`

<sup>3</sup> ENS de Lyon, Université de Lyon

**Abstract.** In this paper, we design the first linear-time algorithm for computing the prime decomposition of a digraph  $G$  with regard to the cartesian product. A remarkable feature of our solution is that it computes the decomposition of  $G$  from the decomposition of its underlying undirected graph, for which there exists a linear-time algorithm. First, this allows our algorithm to remain conceptually very simple and in addition, it provides new insight into the connexions between the directed and undirected versions of cartesian product of graphs.

The general idea of graph decompositions is to describe a graph as the composition, through some operations, of a set of simpler (and usually smaller) graphs. This framework has turned out to be very useful both for proving theorems (see e.g. [2]) and for solving efficiently difficult algorithmic problems using the "divide and conquer" approach (see [10]). This is the reason why, in the last decades, a lot of effort have been made for computing efficiently the decomposition of a graph with respect to a given operation. The cartesian product of undirected graphs (graphs for short) and directed graphs (digraphs for short), usually denoted by  $\square$ , is a classical and useful decomposition operation that allows to factorise some specifically structured redundancy in a graph. Such redundancy naturally appears in various contexts, both theoretic and practical, such as accountability, databases or programming. In those contexts, revealing and factorising these redundancy has a great impact on the efficiency of the solutions proposed to manage these systems.

Cartesian product has been used from the early times of graph theory, but the first intensive studies were provided by Sabidussi [9] and Vizing [11]. Both of them proved independently that any connected graph admits a decomposition into prime factors (graphs that can not be expressed as product of non trivial graphs) which is unique up to the order of factors ( $\square$  is commutative) and up to

---

<sup>\*</sup> This work was partially supported by the Vietnam Institute for Advanced Study in Mathematics (VIASM).

isomorphisms. The unicity of the decomposition into prime factors was later extended to weakly connected digraphs by Feigenbaum [4] and Walker [12] (when graphs or digraphs are not connected, this prime decomposition is not necessarily unique). From the 80s, a series of papers have investigated the complexity of computing this prime decomposition. For connected graphs, after the first polynomial algorithm by Feigenbaum et al [5] in  $O(n^{4.5})$  time, where  $n$  is the number of vertices, the complexity was improved by Winkler [13], Feder [3] and Aurenhammer et al [1], until Imrich and Peterin [7] finally designed a linear-time algorithm (this is so far the only linear algorithm).

**Related Works.** For weakly connected digraphs, the best complexity is achieved by Feigenbaum's algorithm [4] which has two steps: first it calls any algorithm that computes the prime decomposition of the underlying undirected graph (i.e. the graph obtained by replacing directed arcs by undirected edges), then by merging some factors it provides the decomposition for the digraph. The first step can be achieved with Imrich and Peterin's algorithm in time  $O(n + m)$ , where  $m$  is the number of edges. Then the time complexity of the second step is  $O(n^2 \log^2 n)$ . The problem has also been considered for restricted classes of digraphs. For connected partially ordered sets (posets), Walker describes a polynomial algorithm in [12] which also starts by factorising the graph once arcs are made undirected. Its complexity is not analysed precisely in [12], but it is not linear: it has two nested loops leading to, at least, a  $\Theta(m \log^2 n)$  complexity. For sake of completeness, let us mention that, for posets, Krebs and Schmid considered a restricted version of the prime decomposition problem that only asks whether the input poset  $P$  admits a factorisation  $P = P_0 \square \mathbf{2}$ , where  $P_0$  is an arbitrary poset and  $\mathbf{2}$  is the chain with two vertices. In [8], they obtain an  $O(n^7)$  algorithm for solving this problem.

**Our Results.** We present the first linear algorithm to compute the prime decomposition of digraphs with regard to the cartesian product, therefore improving the complexities of [4,12,8]. Unlike [12,8], we solve the problem not only for posets but for arbitrary digraphs  $G$ . An interesting feature of our solution is that we use the algorithm of [7] as a black-box, once, at the beginning of our algorithm. In other words, we first compute the decomposition of the underlying undirected graph  $\tilde{G}$  of  $G$  and afterwards only, we deal with the orientation of arcs, in linear time. This allows our algorithm to remain conceptually very simple. Compared to [4,12], which proceed in the same way, the improvement of the complexity relies on 1) the incremental structure of our algorithm that fully takes advantage of the product structure computed for  $\tilde{G}$  by parsing it layer by layer, and 2) a careful implementation and choice of the data-structure.

**Outline of the Paper.** Section 1 presents the main definitions and theorems that we need about the cartesian product. Section 2 describes the general scheme of our algorithm and the lemmas that will ensure its correctness. Section 3

describes the algorithm and goes into the details of data-structures and routines that are used to run it in linear time.

## 1 Preliminaries

After the formal definition of the cartesian product, we state the factorisation theorem ensuring the unicity of the prime decomposition. This theorem applies to both graphs and digraphs, but it was first proved for graphs [9,11] and then extended to digraphs [4,12]. For any digraph  $G$ , we define its *underlying undirected graph*  $\tilde{G}$  as the graph over the same vertices, where two vertices are adjacent if and only if there exists at least one arc between them in  $G$ . A digraph  $G$  will be called *weakly connected* if  $\tilde{G}$  is connected.

**Definition 1 (Cartesian product of digraphs).** *The cartesian product  $G = \prod_{1 \leq i \leq p} G_i$  of  $p$  directed graphs  $(G_i)_{1 \leq i \leq p}$  is the directed graph  $G = (V(G), A(G))$  whose vertex set is  $V(G) = \prod_{1 \leq i \leq p} V(G_i)$  and such that for all  $x, y \in V(G)$ , with  $x = (x_1, \dots, x_p)$  and  $y = (y_1, \dots, y_p)$ , we have  $xy \in A(G)$  if and only if there exists  $i \in \llbracket 1, p \rrbracket$  such that  $\forall j \in \llbracket 1, p \rrbracket \setminus \{i\}, x_j = y_j$  and  $x_i y_i \in A(G_i)$ . The  $p$ -tuple  $(x_1, \dots, x_p)$  associated with each vertex  $x$  is called the cartesian labelling associated with this decomposition.*

**Definition 2 (Prime graph).** *A digraph  $G$  is prime with regard to the cartesian product iff for all digraphs  $G_1, G_2$  such that  $G = G_1 \square G_2$  then  $G_1$  or  $G_2$  has only one vertex.*

The two preceding definitions are stated for digraphs but they also apply for graphs with edges instead of arcs. We now give the fundamental theorem of cartesian product of graphs.

**Theorem 1 (Unicity of the prime decomposition of digraphs [4,12] and graphs [9,11]).** *For any weakly connected directed graph (resp. connected graph)  $G$ , there exists a unique  $p \geq 1$  and a unique tuple  $(G_1, \dots, G_p)$  of digraphs (resp. graphs), up to reordering and isomorphism of the  $G_i$ 's, such that each  $G_i$  has at least two vertices, each  $G_i$  is prime for the cartesian product and  $G = \prod_{i \in \llbracket 1, p \rrbracket} G_i$ .  $(G_1, \dots, G_p)$  is called the prime decomposition of  $G$ .*

A key property of the prime decomposition, stated by Theorem 2 below, is that it properly refines all other decompositions.

**Definition 3 (Refinement of a decomposition).** *Let  $G$  be a graph or a digraph and let  $(G_1, \dots, G_p)$ , with  $p \geq 1$ , and  $(H_1, \dots, H_k)$ , with  $k \geq 1$ , be two decompositions of  $G$ . We say that decomposition  $(G_1, \dots, G_p)$  refines decomposition  $(H_1, \dots, H_k)$  iff  $k \leq p$  and there exists a partition  $\{I_1, \dots, I_k\}$  of  $\llbracket 1, p \rrbracket$  such that  $\forall j \in \llbracket 1, k \rrbracket, H_j = \prod_{i \in I_j} G_i$ .*

**Theorem 2 (Finest decomposition [6]).** *Let  $G$  be a weakly connected digraph (resp. connected graph) and let  $(G_1, \dots, G_p)$ , with  $p \geq 1$ , be its prime decomposition. If  $(H_1, \dots, H_k)$ , with  $k \geq 1$ , is a decomposition of  $G$  such that all digraphs  $H_i$ 's have at least two vertices, then  $(G_1, \dots, G_p)$  refines  $(H_1, \dots, H_k)$ .*

Cartesian product decomposition of graphs can equivalently been defined as colourings of their arcs or edges. Such colourings constitute the core of the approach of [7], and of our approach as well.

**Definition 4 (Product colouring of arcs (resp. edges) [7]).** Let  $G$  be a digraph (resp. a graph) and  $\mathcal{L}$  the cartesian labelling of a decomposition of  $G$ , then the colouring of arcs of  $G$  associated with  $\mathcal{L}$  is defined as follows: arc (resp. edge)  $xy$  is coloured with colour  $i$  iff  $x$  and  $y$  differ only on coordinate number  $i$ . A colouring of the arcs of a digraph  $G$  (resp. edges of graph) is called a product colouring if it is the colouring associated to some cartesian labelling of some decomposition of  $G$ .

Note that the colouring associated with  $\mathcal{L}$  is properly defined since each arc (resp. edge) is assigned a colour and only one. The next theorem restates Theorem 2 in terms of product colourings. It appears as Lemma 2.3 in [7] for graphs and in [4] for digraphs (stated in terms of partitions of the arcs).

**Theorem 3 (Finest product colouring [7]).** Let  $G$  be a weakly connected digraph (resp. connected graph) and let  $C$  be the arc colouring associated with the prime decomposition of  $G$ , and let  $C'$  be a product colouring of  $G$ . Then, the partition of the arcs of  $G$  induced by  $C$  refines the one induced by  $C'$ .  $C$  is called the finest product colouring of  $G$ .

Product colourings have strong structural properties which are characterised in the next two theorems. These are the key properties on which is based the correctness and the complexity of our algorithm. The first theorem deals with undirected graphs. It rephrases several lemmas and reasoning used by Imrich and Peterin to design and analyse their linear algorithm [7].

**Theorem 4 (Square property of product colourings (undirected version) [7]).** Let  $C$  be a colouring of the edges of some connected graph  $G$ .  $C$  is a product colouring iff the three following properties hold:

1. all triplets of vertices inducing a triangle in  $G$  are monocoloured, and
2. for any bicoloured pair  $\{\{u, v\}, \{v, w\}\}$  of edges of  $G$ , there exists a unique vertex  $v'$  such that  $uvwv'$  is a cycle of  $G$ , and this vertex  $v'$  is such that the colour of  $\{u, v'\}$  (resp.  $\{v', w\}$ ) is the same as the one of  $\{v, w\}$  (resp.  $\{u, v\}$ ).
3. for any colour  $i \in \llbracket 1, |C| \rrbracket$ , if there exists a path of  $G$  from  $x$  to  $y$ , where  $x \neq y$ , made only of edges coloured  $i$ , then there does not exist any path from  $x$  to  $y$  having no edge coloured  $i$ .

*Remark 1.* Due to Condition 1, the cycle  $uvwv'$  in Condition 2 necessarily has no chord: it is called a *square*.

This characterisation can be extended to digraphs, up to adding a fourth (minor) condition and carefully checking the arc orientations in Condition 2 of Theorem 4. For digraphs, a pair of vertices  $x, y$  is called *monocoloured* if all arcs between  $x$  and  $y$  (possibly two) have the same colour. We also define some types to enumerate the different cases of arc orientation between two vertices.

**Definition 5.** Let  $G$  be a digraph, the type of a couple  $(x, y)$  of adjacent vertices, denoted  $\text{type}(x, y)$ , is:  $\text{dir}$  iff  $xy$  is an arc in  $G$  but not  $yx$ ;  $\text{ind}$  iff  $yx$  is an arc in  $G$  but not  $xy$ ; and  $\text{sym}$  iff both  $xy$  and  $yx$  are arcs in  $G$ .

**Theorem 5 (Square property of product colourings (directed version) [7,4]).** Let  $C$  be a colouring of the arcs of some weakly connected digraph  $G$ .  $C$  is a product colouring iff the four following properties hold:

1. all pairs of vertices are monocoloured, and
2. all triplets of vertices inducing a triangle in  $\tilde{G}$  are monocoloured, and
3. for any bicoloured pair  $\{\{u, v\}, \{v, w\}\}$  of edges of  $\tilde{G}$ , there exists a unique vertex  $v'$  such that  $uvwv'$  is a cycle of  $\tilde{G}$ , and this vertex  $v'$  is such that the type and the colour of  $(u, v')$  (resp.  $(v', w)$ ) are the same as those of  $(v, w)$  (resp.  $(u, v)$ ).
4. for any colour  $i \in \llbracket 1, |C| \rrbracket$ , if there exists a path of  $\tilde{G}$  from  $x$  to  $y$ , where  $x \neq y$ , made only of edges coloured  $i$ , then there does not exist any path from  $x$  to  $y$  having no edge coloured  $i$ .

Though not stated in a single theorem in literature, this theorem combines Theorem 4 and the forbidden oriented patterns identified by Feigenbaum in products of digraphs [4]. We take them into account by forcing the types of adjacency around the squares of the digraph in Condition 3 of Theorem 5. The preservation of types on opposite edges of squares is due to the fact that both edges originate from the same pair of vertices in a factor of  $G$ , thus their types exactly reflect the type of this pair. Like for Theorem 4, we have rephrased the characterisation for our needs. We will make an intensive use of Theorem 5 to prove the correctness of our approach.

## 2 Our Approach

Like [7], our approach consists in computing the finest product colouring  $\mathcal{C}_G$  of the arcs of  $G$ , which is precisely the one corresponding to the prime decomposition of  $G$ . We proceed in two steps: i) first, we compute the finest product colouring  $\mathcal{C}_{\tilde{G}}$  of the undirected underlying graph  $\tilde{G}$  of  $G$  and we colour the arcs of  $G$  accordingly ii) then, we merge some classes of colours into one single colour in order to obtain  $\mathcal{C}_G$ . Our main result is to show that the classes of colours to be merged can be computed in linear time with regard to the size of  $G$ , and that the labels of the vertices can be updated in linear time as well during these merges. The fact that one can always proceed by merging some colours of the undirected colouring of  $\tilde{G}$  is stated by Lemma 1 below.

**Lemma 1.** Let  $G$  be a digraph, let  $\mathcal{C}_G$  be the finest product colouring of  $G$ , and let  $\mathcal{C}_{\tilde{G}}$  be the finest product colouring of  $\tilde{G}$ . We denote  $\mathcal{C}_{\text{dir}}$  the colouring of the arcs of  $G$  induced by  $\mathcal{C}_{\tilde{G}}$ . Then,  $\mathcal{C}_{\text{dir}}$  is finer than  $\mathcal{C}_G$ .

*Sketch of proof.* In a directed product colouring, two symmetric arcs are always assigned the same colour. Then, each directed product colouring of  $G$  induces

a colouring of the edges of  $\tilde{G}$ . It turns out that this colouring of  $\tilde{G}$  is also a product colouring, since the conditions to be an undirected product colouring are weaker than the conditions to be a directed product colouring. Now consider the undirected product colouring induced by  $\mathcal{C}_G$ , since it is a product colouring, from Theorem 3, it is coarser than  $\mathcal{C}_{\tilde{G}}$ . It follows that  $\mathcal{C}_G$  is coarser than  $\mathcal{C}_{dir}$ .

In order to design an algorithm, we must be able to determine which classes of colours have to be merged in  $\mathcal{C}_{dir}$  in order to obtain the finest product colouring  $\mathcal{C}_G$  we aim at computing. We will show that  $\mathcal{C}_G$  can be obtained by merging all the pairs of colours that are *conflicting* in  $\mathcal{C}_{dir}$ .

**Definition 6.** Let  $G$  be a digraph and let  $\mathcal{C}$  be a colouring of its arcs such that all pairs of vertices of  $G$  are mono-coloured. Two colours  $c_1, c_2$  of  $\mathcal{C}$  are conflicting iff there exists some bicoloured pair  $\{\{u, v\}, \{v, w\}\}$  of edges of  $\tilde{G}$  such that  $\{u, v\}$  is coloured by  $c_1$  and  $\{v, w\}$  is coloured by  $c_2$  and  $\{\{u, v\}, \{v, w\}\}$  does not satisfy Condition 3 of Theorem 5.

The list of conflicting pairs of colours in  $\mathcal{C}_{dir}$  defines a graph  $G_{conf}$ , which we call the *colour-conflict graph*, whose vertex set is the colours of  $\mathcal{C}_{dir}$ . Lemma 2 below claims that the classes of colours of  $\mathcal{C}_{dir}$  that have to be merged into a single colour in order to obtain the finest product colouring  $\mathcal{C}_G$  of  $G$  are precisely the connected components of  $G_{conf}$ .

**Lemma 2.** Let  $G$  be a digraph and let  $\mathcal{C}_{dir}$  be the colouring of  $G$  induced by the finest product colouring of  $\tilde{G}$ . Consider the colour-conflict graph  $G_{conf}$  whose vertices are the colours of  $\mathcal{C}_{dir}$ .

Then, the colouring  $\mathcal{C}_{conf}$  of the arcs of  $G$  obtained by merging in  $\mathcal{C}_{dir}$  each connected component of  $G_{conf}$  into one single colour is the finest product colouring  $\mathcal{C}_G$  of  $G$ .

*Sketch of proof.* From Lemma 1, we know that  $\mathcal{C}_G$  can be obtained from  $\mathcal{C}_{dir}$  by only merging some colours. If there is a conflict between colours  $c_1$  and  $c_2$  in  $\mathcal{C}_{dir}$  on some pair  $\{\{u, v\}, \{v, w\}\}$ , then, clearly, the only possibility so that the conflict disappear in  $\mathcal{C}_G$  is to merge colours  $c_1$  and  $c_2$ . Thus, all pairs of conflicting colours have to be merged, which results in the merge of all the colours in a connected component of  $G_{conf}$ . Thus,  $\mathcal{C}_{conf}$  is a colouring of the arcs of  $G$  finer than  $\mathcal{C}_G$ . On the other hand, when all these merges have been performed, there is no remaining conflicts between colours, that is Condition 3 of Theorem 5 is satisfied. Consequently, since the other conditions of Theorem 5 are satisfied in  $\mathcal{C}_{dir}$  and since these conditions are preserved by merging colours, it follows that  $\mathcal{C}_{conf}$  satisfies all the conditions of Theorem 5 and is therefore a product colouring of  $G$ . As  $\mathcal{C}_{conf}$  is finer than  $\mathcal{C}_G$ , we have  $\mathcal{C}_{conf} = \mathcal{C}_G$ .

A key property which we will use in the description of our algorithm, and which allows it to remain conceptually simple and to run in linear time, is that the bicoloured pairs of vertices  $\{\{u, v\}, \{v, w\}\}$  giving rise to a conflict on their colours are strongly structured: they necessarily belong to the set of properly

coloured squares of  $\mathcal{C}_{dir}$ . In other words, the only reason why a conflict may appear between two colours of  $\mathcal{C}_{dir}$  is because of the orientation of arcs. This is what is stated by Lemma 3 below.

**Lemma 3.** *Let  $G$  be a digraph, let  $\mathcal{C}_{\tilde{G}}$  be the finest product colouring of  $\tilde{G}$ , and let  $\mathcal{C}_{dir}$  be the colouring of the arcs of  $G$  induced by  $\mathcal{C}_{\tilde{G}}$ . If two colours are conflicting in  $\mathcal{C}_{dir}$  on some bicoloured pair  $\{\{u, v\}, \{v, w\}\}$ , then*

1. *there exists a unique vertex  $v'$  such that  $uvvv'$  is a cycle of  $\tilde{G}$ , and this vertex  $v'$  is such that in  $\mathcal{C}_{dir}$ ,  $\text{colour}(u, v') = \text{colour}(v, w)$  and  $\text{colour}(v', w) = \text{colour}(u, v)$ , but*
2.  *$\text{type}(u, v') \neq \text{type}(v, w)$  or  $\text{type}(v', w) \neq \text{type}(u, v)$ .*

*Proof.* Since pair  $\{\{u, v\}, \{v, w\}\}$  is bicoloured in  $\mathcal{C}_{dir}$  and since  $\mathcal{C}_{dir}$  is induced from  $\mathcal{C}_{\tilde{G}}$ , then pair  $\{\{u, v\}, \{v, w\}\}$  is bicoloured in  $\mathcal{C}_{\tilde{G}}$ . And since  $\mathcal{C}_{\tilde{G}}$  is a product colouring, then, from Condition 2 of Theorem 4, Condition 1 of Lemma 3 is satisfied. But since bicoloured pair  $\{\{u, v\}, \{v, w\}\}$  is conflicting, it does not satisfy Condition 3 of Theorem 5. Thus, necessarily, we have  $\text{type}(u, v') \neq \text{type}(v, w)$  or  $\text{type}(v', w) \neq \text{type}(u, v)$ .

### 3 Algorithm

Our algorithm takes as input the adjacency lists of the digraph  $G$  and outputs the finest product colouring of the arcs of  $G$  together with the corresponding cartesian labelling of the vertices of  $G$ . It operates in three steps:

1. *Undirected prime decomposition:* apply Imrich and Peterin's algorithm [7] on  $\tilde{G}$  and obtain the induced colouring  $\mathcal{C}_{dir}$  of the arcs of  $G$ .
2. *Conflicting pairs of colours:* list all pairs of colours that are conflicting in  $\mathcal{C}_{dir}$  and build the colour-conflict graph  $G_{conf}$ .
3. *Merge:* merge each connected component of  $G_{conf}$  into one single colour and update the labels of the vertices of  $G$  accordingly.

In this section, we deals with implementation details and show how to perform all of the three steps above in linear time with regard to the size of  $G$ .

**Cartesian Representation.** Given a product colouring  $\mathcal{C}$  of a digraph  $G$  (or a product colouring of its underlying undirected graph  $\tilde{G}$ ) and the corresponding cartesian labelling  $V = \prod_{i \in \llbracket 1, p \rrbracket} V_i$  of its vertices, we encode the digraph  $G$ , its colouring  $\mathcal{C}$  and the cartesian labels of its vertices into a data-structure that we call the *cartesian representation* of  $G$ . It is very similar to the classical adjacency lists, except that the lists of neighbours of the vertices are stored in a matrix  $M_{cart}$  instead of a one-dimensional array.

For all  $i \in \llbracket 1, p \rrbracket$ , we denote  $n_i = |V_i|$ . The vertices in  $V_i$  are numbered from 1 to  $n_i$  so that the label  $(x_1, \dots, x_p)$  of a vertex  $x$  belongs to  $\llbracket 1, n_1 \rrbracket \times \dots \times \llbracket 1, n_p \rrbracket$ .  $M_{cart}$  is an  $n_1 \times \dots \times n_p$  matrix indexed by the labels of the vertices of  $G$  and such that the cell  $M_{cart}(x_1, \dots, x_p)$  contains two fields storing the information of the

vertex  $x$  whose label is  $(x_1, \dots, x_p)$ : the first field is simply the label  $(x_1, \dots, x_p)$  of  $x$ , and the second field is a one dimensional array denoted  $N(x)$  and indexed by the  $p$  colours of colouring  $\mathcal{C}$ , from 1 to  $p$ . For any  $i \in [1, p]$ , the cell indexed  $i$  of  $N(x)$  contains the list  $N_i(x)$  of neighbours  $y$  of  $x$  such that the arcs between  $x$  and  $y$  are coloured  $i$ . Each cell of list  $N_i(x)$  corresponding to a neighbour  $y$  of  $x$  again contains two fields: the first one is  $type(x, y)$  and the second one is a pointer to the cell  $M_{cart}(y_1, \dots, y_p)$  of  $M_{cart}$ , where  $(y_1, \dots, y_p)$  is the label of  $y$ . Moreover, in the cartesian representation, each list  $N_i(x)$  is sorted by increasing value of the  $i$ -th component  $y_i$  of the neighbours  $y = (y_1, \dots, y_i, \dots, y_p)$  of  $x$  it contains. Note that since only the component  $y_i$  changes in the list  $N_i(x)$ , the order defined on  $N_i(x)$  by the  $i$ -th component of the label is exactly the cartesian order on the label of vertices of  $N_i(x)$ . Finally, let us mention that the reason why we use a pointer to  $M_{cart}(y_1, \dots, y_p)$  instead of simply the label  $(y_1, \dots, y_p)$  of  $y$  is that reading one pointer takes constant time while reading a sequence of integer takes a time proportional to the length of the sequence. This feature is necessary in order to achieve linear time.

**Undirected Prime Decomposition.** In this step, from the adjacency lists of  $G$ , we compute the cartesian representation of  $G$  with regard to the product colouring  $\mathcal{C}_{\tilde{G}}$  of  $\tilde{G}$  given by the algorithm from [7]. First of all, in order to apply the algorithm from [7], we need to compute the undirected adjacency lists of  $\tilde{G}$ , from the directed lists of  $G$ . This can be done in linear time, and we can determine in the same time the types of all adjacent couples of vertices, which we write into the cells of the adjacency lists.

The adjacency lists of  $\tilde{G}$  are then given as input to [7]'s algorithm, which computes the prime decomposition  $\tilde{G} = \prod_{i \in [1, p]} \tilde{G}_i$  (as usual we denote  $n_i = |V(\tilde{G}_i)|$ ). The algorithm gives the corresponding cartesian labelling of the vertices of  $G$  and the colouring of the edges of  $\tilde{G}$ . More explicitly, it produces a data-structure that, given a vertex, provides its label in constant time and, given an edge, provides its colour in constant time. Using this data-structure, one can build very easily the cartesian representation of  $G$  described above. The only difficulty is to sort all the coloured adjacency lists according to the cartesian labels of the vertices they contain. To that purpose, we use the classical technique to sort adjacency lists of a graph  $G$  w.r.t. a given order  $\sigma$  on the vertices of  $G$  in linear time: 1) initialise a new copy of the adjacency lists with all lists empty and 2) for each vertex  $x$  of  $G$  considered in increasing order w.r.t.  $\sigma$ , parse its list of neighbours (the order of parsing does not matter here) and for each vertex  $y$  encountered, append  $x$  at the end of the list of  $y$  in the new copy of the adjacency lists. Here, we have to take care in addition of the colours of the arcs and of the type of the couples of vertices, which can be done without penalising the complexity. Then, the first step of our algorithm takes linear time.

**Conflicting Pairs of Colours.** This step of the algorithm outputs the list (eventually with repetitions) of all pairs of conflicting colours. This is the most challenging part of the algorithm as the conflicts may occur on any properly coloured square of  $\mathcal{C}_{dir}$ . And it turns out that the number of such squares may

be quadratic, while we aim at achieving a linear complexity. The key point is that we can actually detect all the conflicts between colours without parsing all the squares of  $\mathcal{C}_{dir}$ , but only a certain subset of them, whose size is linear. To that purpose, we take advantage of the product structure computed for  $\tilde{G}$ , which we parse incrementally layer by layer, each layer being processed in linear time.

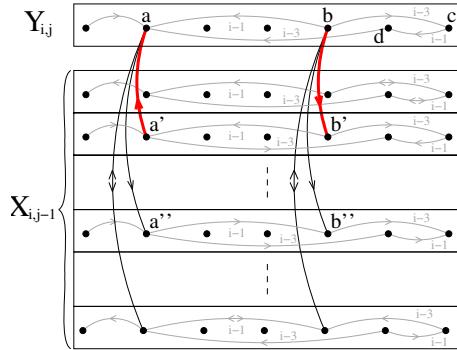
Our algorithm has two passes: the first one computes all the pairs  $(c_1, c_2)$  of colours, with  $c_2 > c_1$ , conflicting on some square because of the orientations of arcs of colour  $c_2$  (remind that from Lemma 3, the only possibility for conflicts to appear in  $\mathcal{C}_{dir}$  is because of the orientation of arcs); then we reverse the order on colours (which can be done easily in linear time on our data-structure) and run the same algorithm. Here, we only describe the first pass.

As previously, we denote  $V = \prod_{i \in [1, p]} V_i$  the cartesian labelling of the vertices of  $G$ , with  $n_i = |V_i|$ . For each  $i \in [1, p]$ , we number the vertices of  $V_i$  from 1 to  $n_i$ . For  $i \in [1, p]$  and  $j \in [1, n_i]$ , we denote  $x_{i,j}$  the vertex of  $V_i$  numbered  $j$  and we denote  $N_{<}(x_{i,j}) = \{x_{i,k} \mid k < j \text{ and } x_{i,j} \text{ and } x_{i,k} \text{ are adjacent in } G_i\}$ , and  $d_{<}(x_{i,j}) = |N_{<}(x_{i,j})|$ . We also denote  $Y_{i,j} = \{x = (x_1, \dots, x_p) \mid x_i = j \text{ and } \forall k > i, x_k = 1\}$ , and  $X_{i,j} = \bigcup_{k \leq j} Y_{i,k}$ . While  $n_{< i}$  and  $m_{< i}$  respectively stand for the number of vertices and the number of adjacent pair of vertices in  $G[Y_{i,j}]$  (these numbers do not depend on  $j$ ).

Our algorithm is incremental in the sense that it starts with the set of vertices  $\{x_{1,j}, j \in [1, n_1]\} \cup \{x_{i,1}, i \in [1, p]\}$ , and considers the vertices  $(x_{i,j})$  of the  $G_i$ 's one by one in increasing  $(i, j)$  for the cartesian order. Before adding  $x_{i,j}$ , it has already computed all the pairs of colours conflicting on some square of  $G[X_{i,j-1}]$ , and when adding  $x_{i,j}$ , our algorithm extends this list with all the pairs  $(c_1, c_2)$  of colours, with  $c_2 > c_1$ , conflicting on some square of  $G[X_{i,j}]$  (because of the orientation of  $c_2$ -coloured arcs) that involve some vertex of  $Y_{i,j}$ . There are two types of such squares (see Figure 1):

1. *layer type*: those involving only vertices of  $Y_{i,j}$ .
2. *cross type*: those involving two vertices  $a, b \in Y_{i,j}$  and two vertices  $a', b' \in X_{i,j-1}$ , which are necessarily, since  $a, b, b', a'$  is a square, such that ( $\forall k \neq i, a'_k = a_k$  and  $b'_k = b_k$ ) and ( $a'_i = b'_i = j'$ ), with  $j' < j$ .

In order to detect the layer-type conflicts, we simply recursively apply the algorithm on  $G[Y_{i,j}]$ . Since the cartesian representation of  $G[Y_{i,j}]$  can be extracted from the one of  $G$  in linear time with regard to the size of  $G[Y_{i,j}]$ , then, in order to show that our algorithm performs in linear time, it is sufficient to show that the cross-type conflicts can be computed in time proportional to the number of arcs incident to vertices of  $Y_{i,j}$ , that is  $O(n_{<i}d_{<}(x_{i,j}) + m_{<i})$  time. Note that for the cross-type conflicts, since we are interested only in the conflicts occurring because of the orientation of arcs of the higher colour, we need only to check the orientations of  $i$ -coloured arcs incident to vertices of  $Y_{i,j}$ . Indeed, in a cross-type square  $a, b, b', a'$ , the arcs between  $a$  and  $a'$  and those between  $b$  and  $b'$  are coloured  $i$ , while the arcs linking  $a$  and  $b$  and linking  $a'$  and  $b'$  have colour at most  $i - 1$ . Note that, from the undirected product structure of  $\mathcal{C}_{dir}$ , for any vertex of  $Y_{i,j}$ , the number of its neighbours in  $X_{i,j-1}$  is  $d_{<}(x_{i,j})$ .



**Fig. 1.** The incremental scheme of our algorithm. The arcs of colours less than  $i$  are depicted in grey, and the arcs of colour  $i$  in black (except  $aa'$  and  $bb'$  that are red and bold). Square  $abcd$  is a layer-type conflicting square, because of the orientations of  $bc$  and  $ad$ . It will be detected during the recursive call of our algorithm on  $G[Y_{i,j}]$ . Square  $abb'a'$  is a cross-type conflicting square, because of the orientations of  $aa'$  and  $bb'$ . This conflict will be detected by our algorithm since  $a$  and  $b$  are linked by an arc and will not be in the same part of partition  $\mathcal{P}$ . On the opposite, the conflict between colour  $i-1$  and  $i$  on the square  $abb''a''$  will not be detected in the first pass of the algorithm, as it occurs because of the orientation of arcs of the lower colour  $i-1$ ; though, it will be detected in the second pass after reversing the order on colours.

In order to list cross-type conflicts, we build an  $n_1 \times \dots \times n_{i-1}$  matrix  $T$  indexed by the vertices  $y$  of  $Y_{i,j}$  and where each cell contains a one-dimensional array  $T(y)$  indexed from 1 to  $d_{<}(x_{i,j})$ . Then, for each vertex  $y \in Y_{i,j}$  we parse the vertices  $z \in N(y) \cap X_{i,j-1}$  in increasing order and we set the corresponding cell of  $T(y)$  to  $type(y, z)$ . This takes  $O(n_{<i}d_{<}(x_{i,j}))$  time. Next, we partition  $Y_{i,j}$  into the classes of vertices  $y$  having the same vector  $T(y)$ , and we label each vertex of  $Y_{i,j}$  with the identifier of the class to which it belongs, that is an integer between 1 and  $n_{<i}$ . We compute this partition  $\mathcal{P}$  by bucket-sorting the vertices of  $y \in Y_{i,j}$  according to the value of  $T(y)$ , with  $T(y)(1)$  as primary key,  $T(y)(2)$  as secondary key, and so on. One pass, for one key, takes  $O(n_{<i})$  time since there are only 3 different values for the key:  $dir$ ,  $ind$  and  $sym$ . Thus, the total cost of the bucket-sort is  $O(n_{<i}d_{<}(x_{i,j}))$ .

The key property on which lean our algorithm is that the colours conflicting with colour  $i$  on some cross-type squares are exactly the colours of the arcs of  $G[Y_{i,j}]$  crossing partition  $\mathcal{P}$ , that is having their extremities in two distinct classes of the partition (see Figure 1 and its caption). Indeed, if a  $j$ -coloured arc between  $a$  and  $b$  crosses the partition, the neighbours  $a'$  and  $b'$  of respectively  $a$  and  $b$  that distinguished the type vectors  $T(a)$  and  $T(b)$  form a conflicting square with  $a$  and  $b$ . And conversely, if there exist a cross-type square  $a, b, b', a'$  involving colour  $j < i$ , then necessarily  $a$  and  $b$  are not in the same part of  $\mathcal{P}$  and the arc between  $a$  and  $b$  is coloured  $j$ .

As a consequence, in order to compute the list of colours conflicting with colour  $i$  on some cross-type square, we simply parse the arcs of  $G[Y_{i,j}]$  and check

whether their two extremities have been assigned the same class identifier of  $\mathcal{P}$ . This takes  $O(m_{<i})$  time and the total time needed to compute the colours conflicting with colour  $i$  on cross-type squares is then  $O(n_{<i}d_{<}(x_{i,j}) + m_{<i})$ . Thus, the running time of our algorithm for listing pairs of colours conflicting in  $G$  is  $O(n + m)$ . Note that the output we get is a list of length  $O(n + m)$  with repetitions and containing at most  $p^2$  distinct pairs of colours. We can thus obtain the list without repetitions by bucket sorting the list according to colours, in  $O(n+m+p^2) = O(n+m)$  time. We can then build the colour-conflict graph  $G_{conf}$  on the set of colours and parse it in order to obtain its connected components, which are the classes of colours we need to merge in  $\mathcal{C}_{dir}$  in order to obtain  $\mathcal{C}_G$  (see Lemma 2). This takes  $O(p^2) = O(m)$  time.

**Merge.** In the previous step, we computed the classes of colours  $C_l, l \in \llbracket 1, q \rrbracket$  that have to be merged in order to obtain the finest product colouring of  $G$ . For any  $l \in \llbracket 1, q \rrbracket$  we denote  $C_l = \{l_1, \dots, l_{|C_l|}\}$ , with  $l_1 < l_2 < \dots < l_{|C_l|}$ . We also denote  $G = \prod_{i \in \llbracket 1, q \rrbracket} G'_i$  the prime decomposition of  $G$ , and we denote  $n'_l = n_{l_1} n_{l_2} \dots n_{l_{|C_l|}}$  the number of vertices of  $G'_l$ . In order to obtain the cartesian representation of  $G$  according to the finest colouring  $\mathcal{C}_G$  of its arcs, we need to achieve three tasks: i) update the labelling of vertices of  $G$  and rearrange the matrix storing the adjacency lists accordingly, ii) for each vertex  $x$  merge its lists of neighbours that now belong to the same class of colours  $C_l$  and iii) sort the obtained lists of neighbours according to the cartesian order on the labels of their vertices. Task ii) is very easy to achieve by simply parsing the arrays  $N(x)$  of the cartesian representation and merge the appropriate lists. For Task iii) we can again use the classical technique that allows to sort adjacency lists in linear time. Therefore, Tasks ii) and iii) need only linear computation time. Let us now focus on Task i).

In order to rearrange the matrix of the cartesian representation according to the new label, we first need to compute some matrices and arrays making the correspondences between ancient and new labels of the vertices and between ancient and new names of colours. First, we number the vertices of the new  $G'_l$ 's as follows. For each  $l \in \llbracket 1, q \rrbracket$ , we build a  $n_{l_1} \times \dots \times n_{l_{|C_l|}}$  matrix  $NewName_l$  where cell  $NewName_l(a_1, \dots, a_{|C_l|})$  contains the rank of  $(a_1, \dots, a_{|C_l|})$  in the cartesian order on  $\llbracket 1, l_1 \rrbracket \times \dots \times \llbracket 1, l_{|C_l|} \rrbracket$ . From matrix  $NewName_l$ , we also compute the converse association array  $AncName_l$  of size  $n'_l$  where cell  $AncName_l(k)$  contains the  $|C_l|$ -tuple  $(a_1, \dots, a_{|C_l|}) \in \llbracket 1, l_1 \rrbracket \times \dots \times \llbracket 1, l_{|C_l|} \rrbracket$  such that  $NewName_l(a_1, \dots, a_{|C_l|}) = k$ . The  $t$ -th component  $a_t$  of  $AncName_l(k)$  is denoted  $AncName_l(k)(t)$ . Finally, for each  $i \in \llbracket 1, p \rrbracket$  we compute the values  $Newcolour(i) = l$ , which is the number  $l$  of the class of colours to which colour  $i$  belongs, and  $Newrank(i)$  which is the rank of  $i$  in the ordered list  $C_l$  of this class of colours. All matrices  $NewName_l$  and arrays  $AncName_l$ , for all  $l$ , as well as arrays  $Newcolour$  and  $Newrank$  can be computed in  $O(np) = O(m)$  time.

Then, we achieve Task i) by building a  $n'_1 \times \dots \times n'_q$  matrix  $M_{new}$  to store the adjacency lists of  $G$  organised according to its prime decomposition, that is the colouring  $\mathcal{C}_G$  of its arcs. Then, for each  $x' = (x'_1, \dots, x'_q) \in \llbracket 1, n'_1 \rrbracket \times \dots \times \llbracket 1, n'_q \rrbracket$

we store in cell  $M_{new}(x'_1, \dots, x'_q)$  the new label  $(x'_1, \dots, x'_q)$  of vertex  $x'$  and the array  $M_{dir}(x_1, \dots, x_p)$  containing the neighbours of vertex  $x'$ , where  $(x_1, \dots, x_p)$  is the former label of vertex  $x'$  in the cartesian representation  $M_{dir}$  (see the *Undirected prime decomposition* step of the algorithm). To that purpose, we only need to compute the  $x_i$ 's, for  $i \in \llbracket 1, p \rrbracket$ , from the  $x'_j$ ,  $j \in \llbracket 1, q \rrbracket$ . This can be done thanks to arrays *Newcolour*, *NewRank* and arrays *AncName<sub>t</sub>* as follows:  $x_i = \text{AncName}_s(x'_s)(t)$  with  $s = \text{Newcolour}(i)$  and  $t = \text{NewRank}(i)$ . For each vertex  $x'$ , writing its new label takes  $O(q)$  time, and computing its former label takes  $O(p)$  time. Then, the total time needed to achieve Task i), including the construction of matrix  $M_{new}$ , is  $O(n + (p + q)n) = O(n + m)$ , which is also the total complexity of the merging step of our algorithm.

As a conclusion, each of the three steps of our algorithm runs in  $O(n + m)$  time, which is then the total complexity of our algorithm for computing the prime decomposition of  $G$ . Within this complexity, our algorithm outputs the corresponding cartesian labelling of vertices of  $G$ , the finest product colouring of the arcs of  $G$ , as well as the cartesian representation of  $G$ , which is a natural data-structure for all algorithms willing to exploit the product structure of  $G$ .

## References

1. Aurenhammer, F., Hagauer, J., Imrich, W.: Cartesian graph factorization at logarithmic cost per edge. *Computational Complexity* 2, 331–349 (1992)
2. Chudnovsky, M., Robertson, N., Seymour, P., Thomas, R.: The strong perfect graph theorem. *Annals of Mathematics* 164(1), 51–229 (2006)
3. Feder, T.: Product graph representations. *Journal of Graph Theory* 16, 467–488 (1992)
4. Feigenbaum, J.: Directed cartesian-product graphs have unique factorizations that can be computed in polynomial time. *Discr. Appl. Math.* 15, 105–110 (1986)
5. Feigenbaum, J., Hershberger, J., Schäffer, A.A.: A polynomial time algorithm for finding the prime factors of cartesian-product graphs. *Discrete Applied Mathematics* 12, 123–138 (1985)
6. Hammack, R., Imrich, W., Klavzar, S.: *Handbook of Product Graphs*. CRC Press (2011)
7. Imrich, W., Peterin, I.: Recognizing cartesian products in linear time. *Discrete Mathematics* 307(3-5), 472–483 (2007)
8. Krebs, M., Schmid, J.: Ordering the order of a distributive lattice by itself. *Journal of Logic and Algebraic Programming* 76, 198–208 (2008)
9. Sabidussi, G.: Graph multiplication. *Mathematische Zeitschrift* 72(1), 446–457 (1960)
10. Spinrad, J.P.: Efficient graph representations. *Fields Institute Monographs*, vol. 19. American Mathematical Society (2003)
11. Vizing, V.G.: The cartesian product of graphs. *Vycisl. Sistemy* 9, 30–43 (1963)
12. Walker, J.W.: Strict refinement for graphs and digraphs. *Journal of Combinatorial Theory Series B* 43(2), 140–150 (1987)
13. Winkler, P.M.: Factoring a graph in polynomial time. *European Journal on Combinatorics* 8, 209–212 (1987)

# Metrical Service Systems with Multiple Servers

Ashish Chiplunkar and Sundar Vishwanathan

Department of Computer Science and Engineering  
Indian Institute of Technology Bombay  
Mumbai, India  
`{ashishc,sundar}@cse.iitb.ac.in`

**Abstract.** The problem of metrical service systems with multiple servers ( $(k, l)$ -MSSMS) proposed by Feuerstein [16] is to service requests, each of which is an  $l$ -point subset of a metric space, using  $k$  servers in an online manner, minimizing the distance traveled by the servers. We prove that Feuerstein’s deterministic algorithm actually achieves an improved competitive ratio of  $k$  ( $\binom{k+l}{l} - 1$ ) on uniform metrics. In the randomized online setting on uniform metrics, we give an algorithm which achieves a competitive ratio  $\mathcal{O}(k^3 \log l)$ , beating the deterministic lower bound of  $\binom{k+l}{l} - 1$ . We prove that any randomized algorithm for MSSMS on uniform metrics must be  $\Omega(\log kl)$ -competitive. On arbitrary metric spaces, we have deterministic lower bounds which are significantly larger than the bound for uniform metrics [8].

For the offline  $(k, l)$ -MSSMS, we give a factor  $l$  pseudo-approximation algorithm using  $kl$  servers on any metric space, and prove a matching hardness result, that a pseudo-approximation using less than  $kl$  servers is unlikely, even on uniform metrics.

**Keywords:**  $k$ -server, metrical service system, online, approximation.

## 1 Introduction

The problem of metrical service systems with multiple servers (MSSMS) generalizes two well-known problems – the  $k$ -server problem [27] and metrical service systems (MSS) [10,27]. These problems share a common paradigm, that there is an underlying metric space and requests are to be served by moving servers on the metric space, in such a way that the total distance traveled by the servers is minimized. For a problem in the online setting, the input is revealed to an algorithm piece by piece, and the algorithm must take irrevocable decisions on seeing each piece. In case of the aforementioned problems, each piece in the input is a request, and the irrevocable decisions are the movements of the servers. A (possibly randomized) online algorithm is said to be  $c$ -competitive if, on every input, it returns a solution whose (expected) cost is at most  $c$  times the cost of optimal solution for that input. The book by Borodin and El-Yaniv [5] gives a nice comprehensive introduction to online algorithms and competitive analysis.

**The  $k$ -Server Problem:** The  $k$ -server problem of Manasse et. al. [27] is, arguably, the most famous among the problems that are naturally posed in the

online setting. The following quote by Koutsoupias, in his beautiful survey on the  $k$ -server problem [24], upholds the importance of this problem.

*The  $k$ -server problem is perhaps the most influential online problem: natural, crisp, with a surprising technical depth that manifests the richness of competitive analysis. The  $k$ -server conjecture, which was posed more than two decades ago when the problem was first studied within the competitive analysis framework, is still open and has been a major driving force for the development of the area online algorithms.*

In the  $k$ -server problem, we have  $k$  servers occupying points in a metric space. Each request is a point in the metric space. To serve the request, one of the servers has to be moved to the requested point. The  $k$ -server conjecture referred to in the quote states that there is a  $k$ -competitive deterministic algorithm for the  $k$ -server problem.

Manasse et. al. [27] proved a lower bound of  $k$  on the competitive ratio of any deterministic algorithm on any metric space with more than  $k$  points. They proved that the competitive ratio is  $k$  only for very specific cases, and posed the  $k$ -server conjecture. The conjecture has been shown to hold for a few metric spaces, for example, the line [9] and tree metric spaces [11]. Fiat, Rabani and Ravid [18] were the first to give an algorithm for the  $k$ -server problem, with competitive ratio bounded by a function of  $k$ , which was later improved by Grove [20,4]. The breakthrough result was due to Koutsoupias and Papadimitriou, who proved that an algorithm, first proposed by Chrobak and Larmore, and called the *Work Function Algorithm* (WFA), is  $(2k-1)$ -competitive [25]. In case of randomized algorithms, the best known lower bound that holds for every metric space is  $\Omega(\log k / \log \log k)$  due to Bartal et. al. [3], and there do exist metric spaces with a lower bound more than  $H_k$  [21]. No better algorithm than the Work Function Algorithm is known, even with randomization. The randomized  $k$ -server conjecture states that there exists a randomized algorithm for the  $k$ -server problem with competitive ratio  $\mathcal{O}(\log k)$  on any metric space. Recent developments, which ingeniously adapt the primal-dual framework to the online setting, have been applied to the  $k$ -server problem, culminating in a  $\mathcal{O}(\log k)$ -competitive randomized algorithm for star metrics [2], and a  $\mathcal{O}(\text{poly log}(k) \text{ poly log}(n))$ -competitive randomized algorithm on metric spaces of  $n$  points [1].

**The Generalized Server Problem:** This is a generalization of the  $k$ -server problem, in which the metric space is a disjoint union of  $k$  metric spaces, mutually separated by an infinite distance. A server is located at one point in each of the subspaces. A request is a set of  $k$  points, one from each subspace. The request is to be served by moving some server to the requested point which lies in its subspace. An interesting problem, called the Weighted Server problem [19] is a particular case of the Generalized Server problem. This problem is same as the  $k$ -server problem, except that the servers have different weights, and the cost of moving a server is equal to the product of its weight and the distance covered. We can thus think of this as the Generalized Server problem where the metric spaces are scaled copies of one another. Fiat and Ricklin [19] were the first to study the Weighted Server problem. They gave a deterministic algorithm

with a competitive ratio of  $2^{2^{\mathcal{O}(k)}}$  for uniform metric spaces. They proved that for every metric space there exist weights so that any deterministic algorithm must have a competitive ratio of  $(k+1)!/2$ . Chrobak and Sgall [13] studied the weighted 2-server problem on uniform spaces, and proved that the Work Function Algorithm achieves the best possible competitive ratio of 5. They proved that in contrast with the  $k$ -server problem, there does not exist a memoryless randomized algorithm with a finite competitive ratio, even for the weighted 2-server problem. Recently Sitters [30] proved that the Work Function Algorithm, in fact, is competitive for the generalized 2-server problem.

**Metrical Service System:** The term Metrical Service System (MSS) was coined by Chrobak and Larmore [12] for the following problem. We have a single server in an underlying metric space. Each request is a set of  $l$  points from the metric space, where  $l$  is called the *width*, and is a parameter to the problem. To serve a request, the server has to be dispatched to one of the requested points.

Finding shortest paths is a fundamental problem in the offline setting. In the online setting it is posed as the problem of Layered Graph Traversal (LGT). This problem, introduced by Papadimitriou and Yannakakis [28], was a precursor of MSS. Fiat et. al. [17] proved that MSS and LGT are in fact equivalent problems. That is, there is a  $c$ -competitive algorithm for MSS if and only if there is a  $c$ -competitive algorithm for LGT. They also proved that there exists a metric space on which the competitive ratio of any deterministic algorithm for the MSS problem is  $\Omega(2^l)$ . Further, they gave an  $\mathcal{O}(9^l)$ -competitive algorithm. They proved that  $l/2$  is a lower bound on the competitive ratio of any randomized algorithm for LGT. Ramesh [29] gave a better deterministic algorithm for LGT, which achieves a competitive ratio of  $l^3 2^l$ , and a randomized  $l^{13}$ -competitive algorithm. He proved that there exists a metric space on which any randomized algorithm must have competitive ratio  $\Omega(l^2/\log^{1+\varepsilon} l)$ , for any  $\varepsilon > 0$ . Burley [7] proved that a variant of the Work Function Algorithm is  $\mathcal{O}(l \cdot 2^l)$ -competitive for the MSS (and hence, the LGT) problem. For the uniform metric space, Chrobak and Larmore [12] proved a lower bound of  $l$  on the competitive ratio of any deterministic algorithm, and also gave an algorithm achieving this bound. It is easily seen that the lower bound holds for any metric space with at least  $l+1$  points.

**Metrical Service System with Multiple Servers:** In a natural generalization of both the  $k$ -server problem and metrical service system, we have  $k$  servers on an underlying metric space, and each request is a set of  $l$  points from the metric space. To serve a request, one of the  $k$  servers has to move to one of the  $l$  requested points. We call this problem Metrical Service System with Multiple Servers, with parameters  $k$  and  $l$  ( $(k, l)$ -MSSMS). It is easy to see that this problem is in fact, a further generalization of the Generalized Server problem. Feuerstein [16] studied this problem for uniform metric spaces and called it the Uniform Service System with parameters  $k$  and  $l$  (USS( $k, l$ )). He proved a lower bound of  $\binom{k+l}{k} - 1$  on the competitive ratio of any deterministic algorithm for this problem. In fact, this proof holds for any (not necessarily uniform) metric space with at least  $k+l$  points. Feuerstein also gave an algorithm and proved

that its competitive ratio is  $k \cdot \min\left(\frac{k^{l+1}-k}{k-1}, \sum_{i=0}^{k-2} l^i + l^k\right)$ . He concluded the paper with the following comment.

*An interesting subject of future research is to extend USS( $k, l$ )<sup>1</sup> to non-uniform metric spaces. This would extend both the work in this paper and the work by Chrobak and Larmore [10] on Metrical Service Systems, where only one server is considered.*

**Our Results:** In Section 2 we present a simple analysis of Feuerstein's algorithm, which improves the bound on its competitive ratio proved in [16], to  $k \cdot \binom{(k+l)}{l} - 1$ . In Section 3 we give a  $\mathcal{O}(k^3 \log l)$ -competitive randomized algorithm on uniform metric spaces, which beats the deterministic lower bound by an exponential factor. In Section 4 we consider the offline  $(k, l)$ -MSSMS problem on arbitrary metric spaces, and we give a pseudo-approximation by a factor of  $l$  using  $kl$  servers. We prove a matching lower bound, assuming the Unique Games Conjecture. We conclude with a number of interesting open problems in Section 5.

## 2 Uniform Metric Spaces: The Hitting Set Algorithm

In this section we analyze the algorithm for  $(k, l)$ -MSSMS on uniform metric spaces given by Feuerstein [16], which he calls the *Hitting Set* algorithm. Feuerstein proved that the competitive ratio of this algorithm is at most  $k \cdot \min\left(\frac{k^{l+1}-k}{k-1}, \sum_{i=0}^{k-2} l^i + l^k\right)$ , whereas we prove an asymptotically better bound,<sup>2</sup> of  $k \cdot \binom{(k+l)}{l} - 1$ .

The Hitting Set (HS) algorithm can be described as follows. HS divides the request sequence into phases, the first phase starting with the first request of the sequence. We say that a request produces a fault whenever the requested set of points is disjoint from the set of points occupied by the servers. Each time a request produces a fault, the algorithm behaves as follows. First, it computes a minimum cardinality set  $H$  of points that intersects all the requests that produced a fault during the current phase. If  $|H| \leq k$  then any  $|H|$  servers are made to occupy all points in  $H$ . Otherwise, if  $|H| > k$ , then the phase terminates and a new phase begins with the current request.

**Theorem 1.** *HS is an  $k \cdot \binom{(k+l)}{l} - 1$ -competitive algorithm for  $(k, l)$ -MSSMS.*

*Proof.* Feuerstein observed that the adversary must incur a cost of at least one per phase. He then proved that at most  $\min\left(\frac{k^{l+1}-k}{k-1}, \sum_{i=0}^{k-2} l^i + l^k\right)$  requests can produce faults in any phase. We improve this upper bound to  $\binom{(k+l)}{l} - 1$ , and our

---

<sup>1</sup> Feuerstein used ‘ $w$ ’ for the width parameter, while we use ‘ $l$ ’.

<sup>2</sup> For instance, when  $k = \Theta(l)$ , Feuerstein’s bound is  $\Omega(l^l)$ , whereas ours is  $\mathcal{O}(c^l)$  for some constant  $c$ .

claim follows, since the algorithm pays at most  $k$  for every request that produces a fault.

Let  $A_1, \dots, A_{r-1}$  be the requests that produced a fault in a given phase, and let  $A_r$  be the first request in the next phase, which must also have produced a fault. Let  $B_i$  be the set of points occupied by the servers when the request  $A_i$  was given. Since  $A_i$  produced a fault, we have for every  $i$  with  $1 \leq i \leq r$ ,  $A_i \cap B_i = \emptyset$ . On the other hand, since no request between  $A_{i-1}$  and  $A_i$  produced a fault, the hitting set chosen to serve  $A_{i-1}$  must be  $B_i$  itself. By the definition of the algorithm, for any  $i, j$  with  $1 \leq j < i \leq r$  we have  $A_j \cap B_i \neq \emptyset$ . The *skew Bollobás theorem* due to Lovász [26] asserts that in any set system  $(A_1, B_1), \dots, (A_r, B_r)$  with  $|A_i| = l$  and  $|B_i| = k$ , which satisfies the two conditions, the number of pairs  $(A_i, B_i)$  is at most  $\binom{k+l}{k}$ . Thus, the number of requests in the phase, that produced a fault, is  $r - 1 \leq \binom{k+l}{k} - 1$ , as required.  $\square$

### 3 Uniform Metric Spaces: Randomized Bounds

In this section we give a randomized version of the Hitting Set algorithm from Section 2, which we call the Randomized Hitting Set (RHS) algorithm, and prove that its competitive ratio is  $\mathcal{O}(k^3 \log l)$ . The algorithm is as follows.

RHS divides the sequence of requests in phases, just like the Hitting Set algorithm. Each time a request produces a fault, the algorithm behaves as follows. First, it computes the minimum cardinality  $s$ , of a set of points that intersects all the requests in the current phase given so far. If  $s \leq k$ , it chooses a set  $H$  uniformly at random from the collection of all the hitting sets of size  $s$ , and then any  $s$  servers are made to occupy all points in  $H$ . Otherwise, if  $s > k$  the current phase ends and a new phase begins with the current request.

It is easily seen that the adversary must incur a cost of at least 1 per phase. We prove that the expected cost of RHS is  $\mathcal{O}(k^3 \log l)$  per phase. Note that the value of  $s$  increases from 1 to  $k$  as the algorithm progresses in a phase. We divide the phase into  $k$  sub-phases, where the  $i^{\text{th}}$  sub-phase is the part of the phase when the value of  $s$  is equal to  $i$ . We will require the following combinatorial lemma.

**Lemma 1.** *Let  $s$  be the size of the smallest hitting set of an  $l$ -uniform set system  $\mathcal{S}$ . Then the number of minimum hitting sets of  $\mathcal{S}$  is at most  $l^s$ .*

We present a proof of this lemma in the full version [8].

**Theorem 2.** *RHS is an  $\mathcal{O}(k^3 \log l)$  competitive algorithm for  $(k, l)$ -MSSMS.*

*Proof.* Consider the  $s^{\text{th}}$  sub-phase of any phase. Let  $A_1, \dots, A_r$  be the sets requested in this sub-phase and let  $B_i$  be the set of locations occupied by the servers, just before the set  $A_i$  was requested. Let  $\mathcal{S}_0$  be the collection of sets requested in the current phase before the  $s^{\text{th}}$  sub-phase started (that is, before  $A_1$  was requested), and let  $\mathcal{S}_i = \mathcal{S}_0 \cup \{A_1, \dots, A_i\}$ . Let  $\mathcal{H}_i$  be the collection of all hitting sets of  $\mathcal{S}_i$ , having size  $s$ . Thus, each  $\mathcal{H}_i$  is an  $s$ -uniform set system, and we have  $\mathcal{S}_0 \subsetneq \mathcal{S}_1 \subsetneq \dots \subsetneq \mathcal{S}_r$  and  $\mathcal{H}_0 \supseteq \mathcal{H}_1 \supseteq \dots \supseteq \mathcal{H}_r$ . Further, if  $B_i \in \mathcal{H}_i$

then  $B_{i+1} = B_i$ , else  $B_{i+1}$  is picked uniformly at random from  $\mathcal{H}_i$ , due to the construction of the algorithm. For each  $B_j$  there exists a unique  $t_j$  such that  $B_j \in \mathcal{H}_{t_j-1} \setminus \mathcal{H}_{t_j}$ .<sup>3</sup> Then either  $t_j > j$  and  $B_{j+1} = B_j$ , or  $t_j = j$  and  $B_{j+1}$  is picked uniformly at random from  $\mathcal{H}_j$ . The number of requests for which the algorithm shifts servers, is thus one less than the size of the set  $T = \{t_j \mid 1 \leq j \leq r\}$ . We next bound the expected size of  $T$ .

Let  $h_i = |\mathcal{H}_i|$ . We know  $B_1 \in \mathcal{H}_0 \setminus \mathcal{H}_1$ , since the current phase started with  $A_1$  which must be disjoint from  $B_1$ . Thus  $1 \in T$  with probability 1. We claim that for  $i > 1$ ,  $\Pr[i \in T] = (h_{i-1} - h_i)/h_{i-1}$ .

For  $j < i$  we say that the event  $E_{ji}$  has occurred if  $i, j \in T$  but no  $i'$  between  $j$  and  $i$  is in  $T$ . This event occurs exactly when  $B_j \notin \mathcal{H}_j$ , that is,  $B_j \cap A_j = \emptyset$ , and the next set  $B_{j+1}$ , chosen uniformly at random from  $\mathcal{H}_j$ , turns out to be in  $\mathcal{H}_{i-1} \setminus \mathcal{H}_i$ . Thus, we have  $\Pr[E_{ji} \mid j \in T] = (h_{i-1} - h_i)/h_j$ . Note that for a fixed  $i$ , the events  $E_{ji}$  are pairwise disjoint and  $i \in T$  if and only if  $E_{ji}$  occurs for some  $j < i$ . Hence  $\Pr[i \in T] = \sum_{j=1}^{i-1} \Pr[E_{ji}] = \sum_{j=1}^{i-1} \Pr[E_{ji} \mid j \in T] \Pr[j \in T] = \sum_{j=1}^{i-1} (h_{i-1} - h_i)/h_j \cdot \Pr[j \in T] = (h_{i-1} - h_i) \sum_{j=1}^{i-1} \Pr[j \in T]/h_j$ . We proceed by induction on  $i$ . As the base case, we have for  $i = 2$ ,  $\Pr[2 \in T] = \Pr[E_{12}] = (h_1 - h_2)/h_1$ , as required. For the inductive step, by induction hypothesis we have

$$\Pr[i \in T] = (h_{i-1} - h_i) \left[ \frac{1}{h_1} + \sum_{j=2}^{i-1} \frac{h_{j-1} - h_j}{h_j h_{j-1}} \right] = \frac{h_{i-1} - h_i}{h_{i-1}}$$

The expected size of  $T$  is thus given by  $\mathbb{E}[|T|] = \sum_{i=1}^r \Pr[i \in T] = 1 + \sum_{i=2}^r (h_{i-1} - h_i)/h_{i-1} \leq \sum_{j=1}^{h_1} 1/j$ . Now,  $h_1 = |\mathcal{H}_1|$  and  $\mathcal{H}_1$  is the collection of all minimum hitting sets of the  $l$ -uniform set system  $\mathcal{S}_1$ , with minimum hitting set size  $s$ . Thus, by Lemma 1,  $h_1 \leq l^s$  and hence  $\mathbb{E}[|T|]$  as well as the expected number of faults in the  $s$ 'th sub-phase, is  $\mathcal{O}(s \log l)$ . The cost incurred, for every request which produced a fault, is at most  $s$ , and hence the total cost is  $\mathcal{O}(s^2 \log l)$ , in the  $s$ 'th sub-phase. Summing over  $s$  from 1 to  $k$ , we infer that the cost incurred in an entire phase is  $\mathcal{O}(k^3 \log l)$ .  $\square$

While we have a  $O(k^3 \log l)$ -competitive algorithm, we also have the following lower bound on the competitive ratio of any randomized online algorithm for  $(k, l)$ -MSSMS on uniform metric spaces.

**Theorem 3.** *The competitive ratio of any randomized online algorithm for  $(k, l)$ -MSSMS against an oblivious adversary<sup>4</sup> is  $\Omega(\log kl)$ .*

We present a proof of this theorem in the full version [8]. In the proof we use a form of Yao's principle [32,6,31] and create a distribution of input sequences which forces every deterministic online algorithm to perform a factor  $\Omega(\log kl)$  worse than the optimum.

<sup>3</sup> Assume  $\mathcal{H}_{r+1} = \emptyset$ .

<sup>4</sup> An oblivious adversary is an adversary who does not have access to the random bits used by the algorithm.

## 4 The Offline Problem

We elaborate on the offline  $(k, l)$ -MSSMS problem in this section. Before that, we briefly describe the offline algorithms for the  $k$ -server problem, and Metrical Service Systems.

The problem of finding the optimal solution to an instance of  $k$ -server problem can be reduced to the problem of finding a min-cost flow on a suitably constructed directed graph [9], and hence the offline  $k$ -server problem can be solved in polynomial time. Note that for any instance of min-cost flow, there exists a solution in which all the flows are integral, and which is no worse than any fractional solution. We will use this observation later. The offline MSS problem can be translated to finding the shortest source-to-sink path in a suitably constructed directed graph of size linear in the size of the instance. Thus, MSS too can be solved in polynomial time. In fact,  $(k, l)$ -MSSMS can be solved in polynomial time for any constant  $k$ . However, the problem is NP-hard for any fixed  $l \geq 2$ , when  $k$  is allowed to vary.

In the subsequent subsections we give a natural Integer Linear Program (ILP) for the offline  $(k, l)$ -MSSMS problem. Although it has an unbounded integrality gap, we use it to design a pseudo-approximation algorithm which uses  $kl$  servers and incurs a cost of at most  $l$  times the optimum. We then prove a lower bound which suggests that nothing better than our pseudo-approximation can be achieved in polynomial time. This result has implications in the online setting also. Feuerstein [16] gave a polynomial time online algorithm for MSSMS on uniform spaces which uses  $kl$  servers, and achieves a competitive ratio of  $kl$  against an adversary using  $k$  servers, where  $l$ , as before, is the size of each request set. Our result implies that it is unlikely, that a polynomial time competitive algorithm using less than  $kl$  servers exists.

### 4.1 ILP Formulation and Pseudo-approximation Algorithm

For the metric space  $(M, d)$  let  $S = \{s^1, \dots, s^k\}$  be the set of initial positions of the  $k$  servers and let the request sequence be  $\rho = (R_1, \dots, R_m)$  where  $R_i \subseteq M$  and  $R_i = \{r_i^1, \dots, r_i^l\}$ . A natural integer linear program is as follows. For each  $1 \leq i < i' \leq m$  and  $1 \leq j, j' \leq l$  we have a variable  $f(i, j, i', j')$ , which is 1 if some server was present at  $r_i^j$  to serve  $R_i$ , and that server was next made to shift to  $r_{i'}^{j'}$  in order to serve  $R_{i'}$ ; and 0 otherwise. For each  $1 \leq k' \leq k$ ,  $1 \leq i \leq m$ ,  $1 \leq j \leq l$  we have a variable  $g(k', i, j)$  which is 1 if the  $k'$ 'th server was first shifted to  $r_i^j$ , to serve  $R_i$ , and 0 otherwise. The objective and the constraints are as follows.

$$\text{Minimize } \sum_{k', i, j} d(s^{k'}, r_i^j) g(k', i, j) + \sum_{i, j, i', j'} d(r_i^j, r_{i'}^{j'}) f(i, j, i', j')$$

For all  $k'$  such that  $1 \leq k' \leq k$

$$\sum_{i, j} g(k', i, j) \leq 1$$

For all  $i, j$  such that  $1 \leq i \leq m, 1 \leq j \leq l$

$$\sum_{k'} g(k', i, j) + \sum_{i' < i, j'} f(i', j', i, j) \geq \sum_{i'' > i, j''} f(i, j, i'', j'')$$

For all  $i$  such that  $1 \leq i \leq m$

$$\sum_{k', j} g(k', i, j) + \sum_{i' < i, j', j} f(i', j', i, j) \geq 1$$

$$f(i, j, i', j') \in \{0, 1\} \text{ for all } i, j, i', j', g(k', i, j) \in \{0, 1\} \text{ for all } k', i, j$$

To relax the above ILP to an LP, we replace the constraints  $f(i, j, i', j') \in \{0, 1\}$  for all  $i, j, i', j'$  and  $g(k', i, j) \in \{0, 1\}$  for all  $k', i, j$  by the constraints  $0 \leq f(i, j, i', j') \leq 1$  for all  $i, j, i', j'$  and  $0 \leq g(k', i, j) \leq 1$  for all  $k', i, j$  respectively. Unfortunately, this relaxation has an unbounded integrality gap [8]. We can, however, round the solution of the LP relaxation to get a pseudo-approximation algorithm for MSSMS.

**Theorem 4.** *There is a polynomial time algorithm, which computes a feasible solution to a given instance of  $(k, l)$ -MSSMS using  $kl$  servers instead of  $k$ , and such that the cost of the solution is at most  $l$  times the cost of the fractional optimal solution of the LP relaxation of the ILP.*

*Proof.* Given an instance of  $(k, l)$ -MSSMS with  $S$ , the set of initial server positions, let the vector  $(f^*, g^*)$  be  $l$  times the optimum (fractional) solution of the LP relaxation. Then we have  $\sum_{k', j} g^*(k', i, j) + \sum_{i' < i, j', j} f^*(i', j', i, j) \geq l$  for each  $i$ . Thus, for each  $i$ , there exists some  $j_i$  such that  $\sum_{k'} g^*(k', i, j_i) + \sum_{i' < i, j'} f^*(i', j', i, j_i) \geq 1$ . Let  $r_i^* = r_i^{j_i}$ . The algorithm solves the LP relaxation and finds  $r_1^*, \dots, r_m^*$ . It then treats  $r_1^*, \dots, r_m^*$  as an instance of the  $kl$ -server problem, and assuming  $l$  servers to be initially located at each point in  $S$ , computes the optimum solution. Note that this is a feasible solution to the given instance of  $(k, l)$ -MSSMS, except that it uses  $kl$  servers instead of  $k$ . To analyze its cost, observe that the vector  $(f^*, g^*)$  gives a (fractional) solution to the instance  $r_1^*, \dots, r_m^*$  of the  $kl$ -server problem. Hence the cost of the optimum solution to this instance is no more than the cost of  $(f^*, g^*)$ , which is in turn,  $l$  times the cost of the optimum fractional solution to the LP relaxation. Thus, the solution returned by the algorithm has cost at most  $l$  times that of the optimum of the LP relaxation.  $\square$

## 4.2 Hardness of Pseudo-approximation

The following hardness result essentially implies that nothing better than what the pseudo-approximation algorithm does, can be achieved in polynomial time.

**Theorem 5.** *Assuming the Unique Games Conjecture (UGC) [22], it is NP-hard to pseudo-approximate the  $(k, l)$ -MSSMS problem, for any fixed  $l \geq 2$ , on the uniform metric space with  $n$  points, within any factor polynomial in  $n$ , using less than  $kl$  servers.*

*Proof.* Suppose there is a polynomial time algorithm  $\mathcal{A}$  which, on a metric space with  $n$  points, can pseudo-approximate the solution of an instance of  $(k, l)$ -MSSMS, using  $K < kl$  servers, within a factor  $f(n) = \mathcal{O}(n^p)$  for some constant  $p$  independent of  $k$ . We use this algorithm to construct a polynomial time algorithm  $\mathcal{B}$  which, when given a  $l$ -uniform hypergraph having a vertex cover of size  $k$ , outputs a vertex cover of the hypergraph of size  $K$ . Unless P=NP, this contradicts the hardness result due to Khot and Regev [23], which states that under the Unique Games Conjecture, it is NP-hard to approximate the size of the vertex cover of  $l$ -uniform hypergraphs within a factor less than  $l$ .

Algorithm  $\mathcal{B}$  does the following. Suppose it is given a hypergraph  $H = (V, E)$  where  $|V| = n$ ,  $E = \{E_1, \dots, E_m\}$ ,  $E_i \subseteq V$   $|E_i| = l$ , and an integer  $k$ , with the promise that  $H$  has a vertex cover of size  $k$ .  $\mathcal{B}$  takes the uniform metric space on the set  $V \uplus W$  where  $W = \{w_1, \dots, w_k\}$ . Then it constructs the instance of  $(k, l)$ -MSSMS, where one server is initially placed at each of the  $w_i$ 's and the request sequence is  $E_1, \dots, E_m$  repeated more than  $kf(n+k)$  (but polynomially many) times. We will call each repetition of the sequence  $E_1, \dots, E_m$  a phase.  $\mathcal{B}$  then uses algorithm  $\mathcal{A}$  to find a pseudo-approximate solution of this instance, using  $K < kl$  servers.

Since the hypergraph has a vertex cover  $V' \subseteq V$  of size  $k$ , the requests can be served by shifting the servers to the points in  $V'$  once for all, at a cost  $k$ . The cost of the solution returned by algorithm  $\mathcal{A}$  is therefore at most  $kf(n+k)$ , which is less than the number of phases. Thus, there exists a phase in which  $\mathcal{A}$  incurs zero cost, which means that  $\mathcal{A}$  does not shift server during this phase at all. Since all the hyperedges are requested in each phase, the set of points occupied by the  $K$  servers in this phase must be a vertex cover of  $H$ .

Having obtained a solution from algorithm  $\mathcal{A}$ ,  $\mathcal{B}$  simply searches for a phase in which  $\mathcal{A}$  incurred zero cost, and returns the set of  $K$  points occupied by the servers during this phase, as an approximate vertex cover of  $H$ . It is easy to see that  $\mathcal{B}$  runs in polynomial time.  $\square$

Theorem 5 can be made independent of the validity of UGC, using the facts that it is NP-hard to approximate the size of the vertex cover of  $l$ -uniform hypergraphs within a factor  $l - 1 - \varepsilon$  [14], and usual graphs within a factor  $1.36 - \varepsilon$  [15]. Theorem 5 implies that unless UGC is false, there does not exist a polynomial time competitive online algorithm for  $(k, l)$ -MSSMS, which uses less than  $kl$  servers. This proves the optimality of Feuerstein's algorithm [16], which runs in polynomial time but uses  $kl$  servers.

## 5 Open Problems

We conclude with a number of interesting problems left open. The most important problem among these is the following.

*Problem 1.* Design an  $f(k, l)$ -competitive deterministic / randomized algorithm for  $(k, l)$ -MSSMS on arbitrary metric spaces, for some function  $f$ .

For  $(k, l)$ -MSSMS on arbitrary metric spaces, we do not have an online algorithm with competitive ratio determined by  $k$  and  $l$  alone. We believe that such an algorithm exists, and the Work Function Algorithm is the prime candidate for the deterministic case. However, it is not known whether the Work Function Algorithm is competitive, even for the Generalized 3-server problem. Not surprisingly, the analysis of this algorithm involves a number of technical complications [8]. We know that if such a deterministic algorithm exists, it must perform a super-polynomial amount of computation on the input. It would be interesting even to find a constant factor competitive algorithm for  $(2, 2)$ -MSSMS on special metric spaces, such as the line.

In the full version of this paper [8] we prove a lower bound of  $\binom{k+2l-1}{k} - \binom{k+l-1}{k}$  on the competitive ratio of any deterministic algorithm for  $(k, l)$ -MSSMS, even when the metric space has only two distances. For arbitrary metric spaces, we also prove a lower bound which is exponential in  $k$  for any fixed  $l$ . These lower bounds are improvements to the bound of  $\binom{k+l}{k} - 1$  in [16]. Both these lower bounds are polynomials in  $l$  for a fixed  $k$ . We believe that a better lower bound exists, which is exponential in  $l$  as well as  $k$ . Such a lower bound can possibly be proved by a construction which combines ideas in the  $(k+1)!/2$  lower bound proof for weighted server problem [19] and those in the  $\Omega(2^l)$  lower bound proof for MSS [17]. In the randomized case, we believe that a lower bound of  $\Omega(\text{poly}(l) \cdot \log k)$  should hold.

*Problem 2.* Prove an exponential (resp  $\Omega(\text{poly}(l) \cdot \log k)$ ) lower bound on the competitive ratio of any deterministic (resp. randomized) algorithm for  $(k, l)$ -MSSMS.

For deterministic algorithms on uniform metric spaces our upper and lower bounds on the competitive ratio differ by the factor  $k$ . We believe that the upper bound can be improved by carefully choosing the hitting sets, in the Hitting Set algorithm, described in Section 2. Similarly, there is a significant gap between the bounds in case of randomized algorithms. In all the upper bound (resp. lower bound) results we have made the conservative assumption that whenever the algorithm (resp. adversary) faults, it can potentially shift all the  $k$  servers and hence the cost incurred is  $k$ , while the adversary (resp. algorithm) shifts at most one server per fault. This assumption introduces a gap of a factor of  $k$  between the bounds.

*Problem 3.* For  $(k, l)$ -MSSMS on uniform metric spaces, close the gap between the lower bound of  $\binom{k+l}{l} - 1$  and the upper bound of  $k \cdot \left( \binom{k+l}{l} - 1 \right)$  in the deterministic case, and the lower bound of  $\Omega(\log kl)$  and the upper bound of  $\mathcal{O}(k^3 \log l)$  in the randomized case.

## References

1. Bansal, N., Buchbinder, N., Madry, A., Naor, J.: A polylogarithmic-competitive algorithm for the  $k$ -server problem. In: IEEE 52nd Annual Symposium on Foundations of Computer Science, pp. 267–276. IEEE (2011)

2. Bansal, N., Buchbinder, N., Naor, J.: A primal-dual randomized algorithm for weighted paging. In: 48th Annual IEEE Symposium on Foundations of Computer Science, pp. 507–517. IEEE Computer Society (2007)
3. Bartal, Y., Bollobás, B., Mendel, M.: A ramsey-type theorem for metric spaces and its applications for metrical task systems and related problems. In: 42nd Annual Symposium on Foundations of Computer Science, pp. 396–405. IEEE Computer Society (2001)
4. Bartal, Y., Grove, E.: The harmonic  $k$ -server algorithm is competitive. *Journal of the ACM* 47(1), 1–15 (2000)
5. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press (1998)
6. Borodin, A., El-Yaniv, R.: On randomization in on-line computation. *Information and Computation* 150(2), 244–267 (1999)
7. Burley, W.R.: Traversing layered graphs using the work function algorithm. *Journal of Algorithms* 20(3), 479–511 (1996)
8. Chiplunkar, A., Vishwanathan, S.: Metrical service systems with multiple servers. CoRR, abs/1206.5392 (2012)
9. Chrobak, M., Karloff, H.J., Payne, T.H., Vishwanathan, S.: New results on server problems. *SIAM Journal on Discrete Mathematics* 4(2), 172–181 (1991)
10. Chrobak, M., Larmore, L.L.: The server problem and on-line games. In: On-Line Algorithms: Proceedings of a DIMACS Workshop. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 7, pp. 11–64 (1992)
11. Chrobak, M., Larmore, L.L.: An optimal on-line algorithm for  $k$ -servers on trees. *SIAM Journal on Computing* 20(1), 144–148 (1991)
12. Chrobak, M., Larmore, L.L.: Metrical service systems: Deterministic strategies. Technical report (1993)
13. Chrobak, M., Sgall, J.: The weighted 2-server problem. *Theoretical Computer Science* 324(2-3), 289–312 (2004)
14. Dinur, I., Guruswami, V., Khot, S., Regev, O.: A new multilayered PCP and the hardness of hypergraph vertex cover. *SIAM Journal on Computing* 34(5), 1129–1146 (2005)
15. Dinur, I., Safra, S.: The importance of being biased. In: Proceedings on 34th Annual ACM Symposium on Theory of Computing, pp. 33–42. ACM (2002)
16. Feuerstein, E.: Uniform Service Systems with  $k$  Servers. In: Lucchesi, C.L., Moura, A.V. (eds.) LATIN 1998. LNCS, vol. 1380, pp. 23–32. Springer, Heidelberg (1998)
17. Fiat, A., Foster, D.P., Karloff, H.J., Rabani, Y., Ravid, Y., Vishwanathan, S.: Competitive algorithms for layered graph traversal. *SIAM Journal on Computing* 28(2), 447–462 (1998)
18. Fiat, A., Rabani, Y., Ravid, Y.: Competitive  $k$ -server algorithms (extended abstract). In: 31st Annual Symposium on Foundations of Computer Science, pp. 454–463. IEEE Computer Society (1990)
19. Fiat, A., Ricklin, M.: Competitive algorithms for the weighted server problem. *Theoretical Computer Science* 130(1), 85–99 (1994)
20. Grove, E.F.: The harmonic online  $k$ -server algorithm is competitive. In: Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, pp. 260–266. ACM (1991)
21. Karlin, A.R., Manasse, M.S., McGeoch, L.A., Owicki, S.S.: Competitive randomized algorithms for nonuniform problems. *Algorithmica* 11(6), 542–571 (1994)
22. Khot, S.: On the power of unique 2-prover 1-round games. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing, pp. 767–775. ACM (2002)

23. Khot, S., Regev, O.: Vertex cover might be hard to approximate to within  $2 - \varepsilon$ . *Journal of Computer and System Sciences* 74(3), 335–349 (2008)
24. Koutsoupias, E.: The  $k$ -server problem. *Computer Science Review* 3(2), 105–118 (2009)
25. Koutsoupias, E., Papadimitriou, C.H.: On the  $k$ -server conjecture. *Journal of the ACM* 42(5), 971–983 (1995)
26. Lovász, L.: Flats in matroids and geometric graphs. In: Proc. Sixth British Combinatorial Conf., Combinatorial Surveys, Royal Holloway Coll., Egham, pp. 45–86. Academic Press, London (1977)
27. Manasse, M.S., McGeoch, L.A., Sleator, D.D.: Competitive algorithms for on-line problems. In: Proceedings of the 20th Annual ACM Symposium on Theory of Computing, pp. 322–333. ACM (1988)
28. Papadimitriou, C.H., Yannakakis, M.: Shortest paths without a map. *Theoretical Computer Science* 84(1), 127–150 (1991)
29. Ramesh, H.: On traversing layered graphs on-line. *Journal of Algorithms* 18(3), 480–512 (1995)
30. Sitters, R.: The generalized work function algorithm is competitive for the generalized 2-server problem. *CoRR*, abs/1110.6600 (2011)
31. Stougie, L., Vestjens, A.P.A.: Randomized algorithms for on-line scheduling problems: how low can't you go? *Operations Research Letters* 30(2), 89–96 (2002)
32. Yao, A.C.-C.: Probabilistic computations: Toward a unified measure of complexity (extended abstract). In: 18th Annual Symposium on Foundations of Computer Science, pp. 222–227. IEEE Computer Society (1977)

# The String Guessing Problem as a Method to Prove Lower Bounds on the Advice Complexity\*

## (Extended Abstract)

Hans-Joachim Böckenhauer, Juraj Hromkovič, Dennis Komm,  
Sacha Krug, Jasmin Smula, and Andreas Srock

Department of Computer Science, ETH Zurich, Switzerland  
`{hjb,juraj.hromkovic,dennis.komm,sacha.krug,  
jasmin.smula,andreas.srock}@inf.ethz.ch`

**Abstract.** The advice complexity of an online problem describes the additional information both necessary and sufficient for online algorithms to compute solutions of a certain quality. In this model, an oracle inspects the input before it is processed by an online algorithm. Depending on the input string, the oracle prepares an advice bit string that is accessed sequentially by the algorithm. The number of advice bits that are read to achieve some specific solution quality can then serve as a fine-grained complexity measure. The main contribution of this paper is to study a powerful method for proving lower bounds on the number of advice bits necessary. To this end, we consider the string guessing problem as a generic online problem and show a lower bound on the number of advice bits needed to obtain a good solution. We use special reductions from string guessing to improve the best known lower bound for the online set cover problem and to give a lower bound on the advice complexity of the online maximum clique problem.

## 1 Introduction

Numerous computational problems work in so-called *online environments*, i. e., frameworks where the input arrives piecewise in successive time steps. An *online algorithm* has to answer every such piece by a part of the final output without knowing anything about the future requests (i. e., the rest of the input). In 1985, Sleator and Tarjan introduced the *competitive ratio* as a tool to measure the quality of such algorithms [21]. For an introduction to online computation and competitive analysis, we refer to the standard literature, e. g., [3].

In this paper, we study the model of *computing with advice* to analyze how much information is necessary and sufficient to enable online algorithms to improve over purely deterministic strategies. The idea is to consider an oracle that sees the whole input in advance and writes binary information about this input onto an *advice tape* that may, at runtime, be accessed by the online algorithm.

---

\* This work is partially funded by the SNF grant 200021–141089.

The idea of online computation with advice was introduced in [11]. Revised versions of this model were simultaneously introduced in [5, 15] and [12]. We follow the most general and exact model from [15] in this paper. The advice complexity was studied for various online problems in, e.g., [2, 4–7, 9, 12, 13, 16, 17, 20].

**Definition 1 (Online Algorithm with Advice [5, 15]).** Let  $I = (x_1, \dots, x_n)$  be an input of an online minimization problem. An online algorithm  $A$  with advice computes the output sequence  $A^\phi(I) = (y_1, \dots, y_n)$  such that  $y_i$  is computed from  $\phi, x_1, \dots, x_i$ , where  $\phi$  is the content of the advice tape, i.e., an infinite binary sequence. For some output sequence  $o$ ,  $\text{cost}(o)$  denotes the cost of  $o$ . An online algorithm  $A$  is  $c$ -competitive with advice complexity  $b(n)$  if there is some non-negative constant  $\alpha$  such that, for every  $n$  and for each input sequence  $I$  of length at most  $n$ , there is some  $\phi$  such that  $\text{cost}(A^\phi(I)) \leq c \cdot \text{cost}(\text{Opt}(I)) + \alpha$  and at most the first  $b(n)$  bits of  $\phi$  have been accessed during the computation of  $A^\phi(I)$ . Here,  $\text{Opt}(I)$  denotes an optimal solution for  $I$ . If  $\alpha = 0$ , then  $A$  is called strictly  $c$ -competitive. Moreover,  $A$  is optimal if it is strictly 1-competitive. The definition for maximization problems is analogous.

The concept of advice complexity enables us to perform a much more fine-grained analysis of the hardness of online problems than using the classical competitive analysis. We are especially interested in *lower bounds* on the advice complexity. Such lower bounds not only tell us something about the information content [15] of online problems, but they also carry over to a randomized setting where they imply lower bounds on the number of random decisions needed to compute a good solution [16]. However, similar to most other computing models, also lower bounds on the advice complexity are hard to prove. Thus, it is desirable to have some generic proof methods to establish lower bounds. In this paper, we approach this goal by considering a generic online problem and showing how to transfer lower bounds on its advice complexity to lower bounds for other online problems.

## 1.1 Our Contribution and Related Work

We study the *string guessing problem* with respect to its advice complexity in Section 2. This problem is shown to be generic with respect to proving lower bounds on the advice complexity. Here, a string of length  $n$  over an alphabet of size  $q$  has to be guessed symbol by symbol. More specifically, we define two versions of the problem where, in the first case, the algorithm gets immediate feedback which decisions would have been correct up to the current time step, whereas, in the second case, this feedback is not supplied. First, we prove a lower bound on the advice necessary to achieve some specific number of correct guesses for both versions. Second, we analyze an upper bound on the size of the advice depending on both  $n$  and  $q$ .

Employing this result, we use reductions from the string guessing problem as a technique to prove lower bounds for other well-studied online problems. It seems to be a promising approach to take further steps in this direction and

develop general methods to prove lower bounds on the advice complexity. Our first application, stated in Section 3, deals with an online version of the set cover problem as introduced in [1]. We show how to use the results on the string guessing problem to give a lower bound that closes an exponential gap between the lower and upper bounds given in [17]. Second, in Section 4, we study an online version of the maximum clique problem where the vertices of the graph arrive consecutively. In every time step, an online algorithm has to decide whether the current vertex is part of the solution or not. We give a lower bound on the number of necessary advice bits that is linear in the number of vertices. Due to space limitations, some of the proofs have been omitted.

Note that Emek et al. [12] followed a similar approach to prove a lower bound on the advice complexity of randomized online algorithms with advice for metrical task systems. They defined a problem called *generalized matching pennies* (GMP), which is basically our string guessing problem with known history, but with a different, somewhat artificial cost function. This cost function depends on a parameter  $\tau > 0$  such that any online algorithm for GMP pays at least  $1/\tau$  in every time step. Any reduction from GMP depends on the concrete choice of  $\tau$ . Moreover, their proof method only gives meaningful results for a number  $b$  of advice bits per time step that is bounded by  $1 \leq b \leq \frac{1}{3} \log_2 q$ , whereas our results cover the complete range  $0 \leq b \leq \log_2 q$ . (Note that  $\log_2 q$  advice bits per time step obviously allow an algorithm to be optimal.) In particular, in our model it is possible to prove sublinear upper bounds due to the fact that also values of  $b \in o(1)$  are allowed. Additionally, our model allows to prove lower bounds for any alphabet size  $q \geq 2$  and for any desired competitive ratio, whereas their lower bounds are limited to alphabets of size at least 4 and competitive ratios strictly less than 2. Overall, our string guessing problem offers a more general, uniform, easy-to-use technique to prove lower bounds on the advice complexity of online problems.

## 2 The String Guessing Problem

In many online problems, the question arises whether knowing the history, i. e., the parts of an optimal solution that correspond to the input known at a specific time step, has an effect on the additional information necessary to achieve a certain competitive ratio. We deal with this question studying a very generic online problem, namely the string guessing problem. In the first variant, the algorithm has to guess a character from some fixed alphabet, then, in the next step, it is told what would have been the correct answer and is asked for the next character. In the second variant, the algorithm also has to guess character by character, but it gets no feedback about whether its answer was correct or not until the very end of the request sequence. In both cases, the length  $n$  of the string is given as the first request and the algorithm then has to guess  $n$  characters step by step.

In what follows, the *Hamming distance* between two strings of the same length denotes the number of positions at which these two strings differ. Let us begin by defining the two variants of the string guessing problem formally.

**Definition 2 (String Guessing with Known History).** *The string guessing problem with known history over an alphabet  $\Sigma$  of size  $q \geq 2$  ( $q$ -SGKH) is the following online minimization problem. The input  $I = (n, d_1, d_2, \dots, d_n)$  consists of a natural number  $n$  and the characters  $d_1, d_2, \dots, d_n \in \Sigma$ , that are revealed one by one. The online algorithm  $\mathbf{A}$  computes the output sequence  $\mathbf{A}(I) = y_1 y_2 \dots y_n$ , where  $y_i = f(n, d_1, \dots, d_{i-1}) \in \Sigma$ , for some computable function  $f$ . The algorithm is not required to respond with any output in the last time step. The cost of a solution  $\mathbf{A}(I)$  is the number of wrongly guessed characters, i. e., the Hamming distance  $\text{Ham}(d, \mathbf{A}(I))$  between  $d = d_1 d_2 \dots d_n$  and  $\mathbf{A}(I)$ .*

**Definition 3 (String Guessing with Unknown History).** *The string guessing problem with unknown history over an alphabet  $\Sigma$  of size  $q \geq 2$  ( $q$ -SGUH) is the following online minimization problem. The input  $I = (n, ?, ?, ?, \dots, ?, d)$  consists of the input size  $n$  in the first request,  $n-1$  subsequent requests “?” carrying no extra information, and the correct string  $d = d_1 \dots d_n \in \Sigma^n$ . In each of the first  $n$  time steps, the online algorithm  $\mathbf{A}$  is required to output one character from  $\Sigma$ , forming the output sequence  $\mathbf{A}(I) = y_1 y_2 \dots y_n$ . As above, the algorithm is not required to respond with any output in the last time step, where the string  $d$  is revealed. The cost of a solution  $\mathbf{A}(I)$  is again the Hamming distance between  $d$  and  $\mathbf{A}(I)$ .*

For simplicity, we sometimes speak about the input string  $d = d_1 d_2 \dots d_n$  when we mean the input sequence  $I = (n, d_1, d_2, \dots, d_n)$  or  $I = (n, ?, ?, ?, \dots, ?, d)$  with  $n = |d|$ . Also, we write  $\mathbf{A}(d)$  instead of  $\mathbf{A}(I)$ . Since the cost of an optimal solution for any string guessing instance is always 0, it is not meaningful to consider the competitive ratio as a measure for these problems. We therefore restrict our analysis to the number of errors produced by an algorithm. Our goal is to minimize this number.

It is easy to see that, for every deterministic online algorithm without advice, there is an input string of length  $n$  such that the algorithm produces  $n$  errors. This holds for any alphabet of arbitrary size  $q \geq 2$ . To see this, consider an adversary  $\mathbf{Adv}$  that, in each time step, produces an input character  $\alpha_i$  differing from the deterministic output  $y_i = f(n, \alpha_1, \dots, \alpha_{i-1})$  of the algorithm. Clearly, no deterministic online algorithm gains anything by knowing the history.

Considering online algorithms with advice that produce optimal solutions, we easily see that each such algorithm needs to read at least  $\lceil n \log_2 q \rceil$  advice bits to be optimal on any input of length  $n$ , even in the case of known history: Assume an optimal algorithm  $\mathbf{A}$  reads  $m < \lceil n \log_2 q \rceil$  advice bits. There are  $q^n$  possible different input strings, but only  $2^m \leq 2^{\lceil n \log_2 q \rceil - 1} < 2^{n \log_2 q} = q^n$  different advice strings. Thus, at least two different input strings get the same advice. There is one position where these two strings differ for the first time. The algorithm  $\mathbf{A}$  makes a deterministic decision in the corresponding time step that is optimal for at most one of the two solutions and  $\mathbf{Adv}$  can always choose the other one. A matching upper bound, even in the case of  $q$ -SGUH, can be achieved by simply enumerating all possible inputs and encoding the index of the concrete instance using  $\lceil n \log_2 q \rceil$  advice bits.

On the other hand, a constant amount of advice can already help to guess a linear number of characters correctly, even without considering the history.

**Observation 1.** *There is an online algorithm for  $q$ -SGUH that guesses at least  $\lceil n/q \rceil$  positions correctly on any input string of size  $n$  using  $\lceil \log_2 q \rceil$  advice bits.*

*Proof.* In every input string of length  $n$  over an alphabet of size  $q$ , at least one character  $z$  occurs at least  $\lceil n/q \rceil$  times, and it can be specified by the oracle using  $\lceil \log_2 q \rceil$  bits. An online algorithm that outputs  $z$  in every step guesses at least  $\lceil n/q \rceil$  positions correctly.  $\square$

In the remainder of this section, we estimate the number of advice bits necessary and sufficient to obtain a specific cost.

## 2.1 Lower Bounds

First, we show a lower bound on the number of advice bits necessary to guarantee that there is at most a specific number of wrong answers for  $q$ -SGUH. Consider an online algorithm  $A$  using  $b$  advice bits. This can be seen as a collection of  $2^b$  different deterministic algorithms [17]. As all inputs look the same on the first  $n$  requests, the behavior of these algorithms can only depend on the advice.

For each of the  $q^n$  possible inputs, the oracle can choose between  $2^b$  different algorithms, each of which produces a fixed output string. The oracle has to select a set of  $2^b$  such strings, which we call *center strings*, in such a way that the maximum distance of any input string to the nearest of these center strings is minimized. This is exactly the task of constructing a so-called *covering code*. A covering code  $K_q(n, r)$  over an alphabet  $\Sigma$  of size  $q$  of the strings of length  $n$  with radius  $r$  is defined as a set of codewords (elements of  $\Sigma^n$ ) with the property that every string in  $\Sigma^n$  has a distance of at most  $r$  to at least one codeword in  $K_q(n, r)$ . For an overview of covering codes, we recommend [8]. The minimum size of a covering code  $K_q(n, r)$  gives us the number of different advice strings we need to make sure that the worst-case error over all inputs for  $q$ -SGUH is at most  $r$ .

To get a simple lower bound on the size of a covering code  $K_q(n, r)$ , we consider the Hamming balls of radius  $r$  around the center strings. A Hamming ball of radius  $r$  around a string  $s$  in  $\Sigma^n$  consists of all strings  $t$  with Hamming distance  $\text{Ham}(s, t) \leq r$ . Due to symmetry<sup>1</sup> of the hypercube, the size of a Hamming ball of radius  $r$  around some string  $s$  does not depend on  $s$ . Thus, we denote it by  $\text{Vol}_q(n, r)$ . To make sure that no error greater than  $r$  occurs for any input string, the number  $b$  of necessary advice bits has to satisfy the condition

$$q^n \leq 2^b \cdot \text{Vol}_q(n, r) = 2^b \cdot \sum_{i=0}^r \binom{n}{i} (q-1)^i. \quad (1)$$

Let  $H_q(p) = p \log_q(q-1) - p \log_q p - (1-p) \log_q(1-p)$  denote the  $q$ -ary entropy function, for  $0 < p < 1$ . With this,  $\text{Vol}_q(n, r)$  can be estimated as follows.

---

<sup>1</sup> The  $q^n$  strings of length  $n$  over an alphabet of size  $q$  can be seen as the vertices of a  $q$ -ary hypercube, which is known to be vertex-symmetric.

**Lemma 1 (Guruswami et al. [14]).** Let  $p \in \mathbb{R}$ ,  $q \in \mathbb{N}^{>1}$ ,  $0 < p \leq 1 - 1/q$ . For sufficiently large  $n$ , we obtain  $\text{Vol}_q(n, pn) \leq q^{H_q(p)n}$ .  $\square$

An easy calculation immediately yields

$$\text{Vol}_q(n, pn) \leq q^{H_q(p)n} = \left(\frac{q-1}{p}\right)^{pn} \left(\frac{1}{1-p}\right)^{(1-p)n}. \quad (2)$$

This observation leads to the following linear lower bound for  $q$ -SGUH.

**Theorem 1.** Consider an input string of length  $n$  for  $q$ -SGUH, for some  $n \in \mathbb{N}$ . The minimum number of advice bits for any online algorithm that is correct for more than  $\alpha n$  characters, for  $1/q \leq \alpha < 1$ , is  $(1 - H_q(1 - \alpha)) n \log_2 q$ .

The above argument heavily relies on the fact that, in case of  $q$ -SGUH, the output of a deterministic algorithm is unambiguously determined by the given advice. In the case of  $q$ -SGKH, this is no longer true. A deterministic algorithm might base its output on the history and thus might output different strings while reading the same advice. In the following we show that, despite of this complication, the same lower bound as in Theorem 1 also holds for  $q$ -SGKH.

For the analysis, we use the  $q$ -ary tree  $T_n$  of depth  $n$  as a representation of the set  $\Sigma^n$  of all input strings of length  $n$  over the alphabet  $\Sigma$  (see Fig. 1). For  $0 \leq i \leq q^n - 1$ , the leaf  $v_{(0,i)}$  represents the  $i$ th string in lexicographic order in  $\Sigma^n$  and every inner vertex  $v_{(h,i)}$ , for  $1 \leq h \leq n$ , represents all  $2^h$  strings of the leaves of the subtree rooted in  $v_{(h,i)}$ .

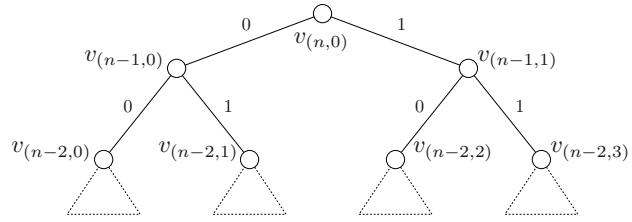


Fig. 1.  $T_n$  representing all input instances for  $q = 2$

Let  $A$  be an online algorithm for  $q$ -SGKH that uses at most  $b$  advice bits for any input instance of length  $n$ . Due to the pigeonhole principle, at least one advice string is used for at least  $\lceil q^n/2^b \rceil$  different input instances. For a given advice string  $\phi$  of length  $b$ , we now take a closer look at the set  $\mathcal{I}_\phi$  of input strings for which  $A$  gets the advice string  $\phi$ . The algorithm  $A$  is not able to distinguish between any two strings in  $\mathcal{I}_\phi$  at the beginning of the computation. However, this situation may change during the computation since  $A$  gets the additional information of what would have been the correct output in every time step so far.

We now investigate the maximal cardinality of  $\mathcal{I}_\phi$  such that  $A$  can guarantee that the maximal number of errors is  $r$ . We can view every computation of  $A$  as a path in  $T_n$  from the root down to a leaf. In every time step,  $A$  decides which subtree to enter. In the following step, it is revealed which direction would have been correct. If instances in more than one subtree of some vertex are represented by the given advice,  $A$  cannot know which subtree is correct. Thus, the adversary can enforce an error in this step by selecting the subtree not chosen by  $A$ .

For any vertex  $v$  in  $T_n$ , let  $F(v)$  denote the maximal number of errors the adversary  $\text{Adv}$  can enforce in the remaining input string inside the subtree rooted at  $v$ , in addition to the errors already made on the way from the root to  $v$ . Moreover, let  $\Psi: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  be a function such that  $\Psi(h, r)$  measures how many strings in  $\mathcal{I}_\phi$  can at most be represented by a vertex at depth  $h$  such that the enforceable number of errors is at most  $r$ . We are interested in the function  $\Psi(n, r)$ , which gives us the desired lower bound and can be computed as follows.

**Lemma 2.** *For  $0 \leq r \leq h \leq n$ , we have  $\Psi(h, r) = \sum_{i=0}^r \binom{h}{i} (q-1)^i = \text{Vol}_q(h, r)$ .*

From Lemma 2 for  $h = n$  together with Lemma 1, we immediately get the following lower bound on the advice complexity for  $q$ -SGKH. Note that this result coincides with the above lower bound for  $q$ -SGUH.

**Theorem 2.** *Consider an input string of length  $n$  for  $q$ -SGKH, for some  $n \in \mathbb{N}$ . The minimum number of advice bits for any online algorithm that is correct for more than  $\alpha n$  characters, for  $1/q \leq \alpha < 1$ , is  $(1 - H_q(1 - \alpha)) n \log_2 q$ .  $\square$*

For  $q = 2$ , i.e., for the bit string guessing problem, we get the following result.

**Corollary 1.** *Consider as input a bit string of length  $n$  for 2-SGKH. Every deterministic algorithm that can guarantee to be correct for more than  $\alpha n$  bits, for  $1/2 \leq \alpha < 1$ , needs to read at least  $(1 + (1 - \alpha) \log_2(1 - \alpha) + \alpha \log_2 \alpha) n$  advice bits.  $\square$*

## 2.2 Upper Bounds

To give an upper bound on the advice complexity of  $q$ -SGUH on strings of length  $n$  with at most  $r$  errors, we analyze the minimal size of a covering code of length  $n$  with radius  $r$ .

**Lemma 3 (Moser and Scheder [19]).** *Let  $n \in \mathbb{N}^{>0}$ ,  $q \in \mathbb{N}^{>1}$ ,  $r \in \mathbb{N}$ . Over any alphabet of size  $q$ , there is a covering code of length  $n$  with radius  $r$  of size at most  $\lceil (n \cdot \ln q \cdot q^n) / \text{Vol}_q(n, r) \rceil$ .  $\square$*

To estimate an upper bound on the length of the advice string that is sufficient to guarantee a certain number of correct characters, we need a lower bound on the volume of the Hamming ball of radius  $r$ .

**Lemma 4.** *Let  $p \in \mathbb{R}$ ,  $q \in \mathbb{N}^{>1}$ ,  $0 < p \leq 1 - 1/q$ , such that  $pn \in \mathbb{N}$ . For sufficiently large  $n$ ,  $\text{Vol}_q(n, pn) \geq q^{H_q(p) \cdot n - \frac{1}{2} \log_q(2n)}$ .*

Now we are ready to prove a linear upper bound on the number of advice bits sufficient to guarantee more than  $\alpha n$  correctly guessed characters.

**Theorem 3.** *Consider an input of length  $n$  for  $q$ -SGUH, for some  $n \in \mathbb{N}$ . There is an online algorithm that is correct for more than  $\alpha n$  characters, for  $1/q \leq \alpha < 1$ , and needs at most  $\lceil (1 - H_q(1 - \alpha)) n \log_2 q + (3 \log_2 n)/2 + \log_2(\ln q) + 1/2 \rceil$  advice bits.*

### 3 The Online Set Cover Problem

In this section, we study the advice complexity of the unweighted online set cover problem, which was introduced in [1] and is defined as follows.

**Definition 4 (Online Set Cover Problem).** Let  $X = \{1, \dots, n\}$  be a ground set of size  $n$ ,  $X' \subseteq X$  a set of requests, and  $\mathcal{S} \subseteq \mathcal{P}(X)$  a set family of size  $m$ . Without loss of generality, no set in  $\mathcal{S}$  is the subset of another set in  $\mathcal{S}$ . The set  $X$  and the family  $\mathcal{S}$  are known beforehand, but the elements of  $X'$  arrive one by one in consecutive time steps. Thus, each permutation of  $X'$  corresponds to one instance of the online set cover problem (SETCOVER). Any subset  $\{S_1, \dots, S_k\}$  of  $\mathcal{S}$  such that  $\bigcup_{i=1}^k S_i \supseteq X'$  is a feasible solution for such an instance. The aim is to minimize  $k$ , i. e., to use as few sets as possible. An online algorithm solves SETCOVER if, immediately after each yet uncovered request  $j$ , it specifies a set  $S_i \in \mathcal{S}$  such that  $j \in S_i$ .

First, we observe that there is a reduction from  $q$ -SGKH to SETCOVER.

**Theorem 4.** Assume an online algorithm A solves SETCOVER with  $b$  advice bits making at most  $r$  errors, i. e., choosing at most  $r$  sets more than an optimal algorithm. Then, there is an online algorithm B that solves the string guessing problem with known history with  $b$  advice bits and at most  $r$  errors.

*Proof sketch.* First, we show how to transform an instance  $I_B$  for  $q$ -SGKH over an alphabet  $\Sigma$  of size  $q$  into an instance  $I_A$  for SETCOVER. Let the considered instance for  $q$ -SGKH be  $I_B = (k, d_1, \dots, d_k)$ . Hence, the input string  $d = d_1 \dots d_k$  has length  $k$ . We describe the set  $X$  and the family  $\mathcal{S}$ , which are known to the SETCOVER algorithm, depending on  $k$  and  $\Sigma$ . We set  $X = \{X_t \mid t \text{ is a string over } \Sigma \text{ of length at most } k\}$ . Then,  $X$  contains  $\sum_{i=0}^k q^i = (q^{k+1} - 1)/(q - 1)$  elements. Moreover,  $\mathcal{S} = \{\text{tr}(s) \mid s = s_1 \dots s_k \in \Sigma^k\}$ , where  $\text{tr}(s) = \{X_t \mid t \text{ is a prefix of } s\} = \{X_\lambda, X_{s_1}, X_{s_1 s_2}, \dots, X_{s_1 \dots s_k}\}$  is the transformation of  $s$  and  $\lambda$  denotes the empty string. Each set  $\text{tr}(s)$  contains  $k+1$  elements, and  $\mathcal{S}$  consists of  $q^k$  sets. The requests  $X_\lambda, X_{d_1}, \dots, X_{d_1 \dots d_k}$  correspond to all prefixes of the string  $d_1 \dots d_k$ . Clearly, an optimal solution only uses the set  $\text{tr}(d)$ .

Then we show that, in each time step  $j$ , we can transform the  $j$ th request of the  $q$ -SGKH instance into a request of the SETCOVER instance, and we can also transform the output of a SETCOVER algorithm A into an output of a  $q$ -SGKH algorithm B such that the following holds: If A uses  $b$  advice bits and makes at most  $r$  errors, then also B uses only  $b$  advice bits and makes at most  $r$  errors.  $\square$

The reduction above helps us to establish a lower bound on the advice complexity of SETCOVER. First, we show that it is equally hard to guess a percentage of  $\alpha$  characters over one string of length  $rk$  as to guess the same percentage over  $r$  strings of length  $k$  over an alphabet of the same size. We start by formally defining the problem of guessing  $r$  strings of size  $k$ .

**Definition 5 (( $q, \ell, k$ )-Multiple String Guessing with Known History).** The  $(q, \ell, k)$ -multiple string guessing problem with known history over an alphabet  $\Sigma$  of size  $q \geq 2$  ( $(q, \ell, k)$ -MULTISGKH for short) is to solve  $\ell$  instances

$I_1, \dots, I_\ell$  of  $q$ -SGKH of length  $k$  over an alphabet of size  $q$ . The input  $I$  is the concatenation of  $I_1, \dots, I_\ell$ . The cost of a solution  $\mathbf{A}(I)$  is the sum of the costs on the  $\ell$  consecutive  $q$ -SGKH instances, i.e.,  $\text{cost}(\mathbf{A}(I)) = \sum_{i=1}^{\ell} \text{cost}(\mathbf{A}(I_i))$ .

**Lemma 5.** *Assume an online algorithm  $\mathbf{A}$  solves  $(q, \ell, k)$ -MULTISGKH with  $b$  advice bits and makes  $r$  errors. Then there is also an online algorithm  $\mathbf{B}$  for  $q$ -SGKH on strings of length  $\ell k$  using  $b$  advice bits and making  $r$  errors.*

**Theorem 5.** *Let  $q \in \mathbb{N}^{\geq 2}$ . For any  $k \in \mathbb{N}^{\geq 1}$  and any  $c \in \mathbb{R}^{>1}$ ,  $c \leq 1 + k(1 - 1/q)$ , every  $c$ -competitive online algorithm for SETCOVER needs to read at least  $(1 - H_q(\frac{c-1}{k})) \frac{k \cdot \log_2 q}{q^k} \cdot m$  or  $(1 - H_q(\frac{c-1}{k})) \frac{k \cdot (q-1) \cdot \log_2 q}{q^{k+1} - 1} \cdot n$  advice bits, where  $m = |\mathcal{S}|$  and  $n = |X|$ .*

*Proof.* We give a reduction from  $(q, \ell, k)$ -MULTISGKH. Consider a set  $\{s_1, \dots, s_\ell\}$  of  $\ell$  strings of length  $k$  each over an alphabet of size  $q$  and the corresponding instance  $I = (I_{s_1}, \dots, I_{s_\ell})$  of  $(q, \ell, k)$ -MULTISGKH. We use the construction from the proof of Theorem 4 to construct  $\ell$  SETCOVER instances  $(X_j, \mathcal{S}_j)$  from  $I_{s_j}$  such that the sets  $X_j$  (and thus also the families  $\mathcal{S}_j$ ) are pairwise disjoint. Then we join these subinstances  $(X_j, \mathcal{S}_j)$  to get a SETCOVER instance  $(X, \mathcal{S})$  by setting  $X = \bigcup_{i=1}^{\ell} X_i$  and  $\mathcal{S} = \bigcup_{i=1}^{\ell} \mathcal{S}_i$ . For each  $I_{s_j}$ , we construct a sequence  $I'_{s_j}$  of requests for SETCOVER using the transformation from the proof of Theorem 4. The order of the  $I'_{s_j}$  follows an arbitrary order of the  $I_{s_j}$ , say  $I_{s_1}, \dots, I_{s_\ell}$ .

The constructed SETCOVER instance has an optimal solution of size  $\ell$  since every subinstance has a solution of size 1 and all subinstances are disjoint. The size of the ground set is  $(q^{k+1} - 1)/(q - 1) \cdot \ell = n$  and the size of the set family is  $q^k \cdot \ell = m$ . We know from Theorem 4 that, for each algorithm for SETCOVER that reads  $b$  advice bits and makes  $r$  errors, there is an algorithm  $\mathbf{B}$  for  $q$ -SGKH using the same advice and making the same number of errors. Consider an algorithm  $\mathbf{A}$  for the constructed SETCOVER instance that defines  $\ell$  algorithms  $\mathbf{A}_1, \dots, \mathbf{A}_\ell$  for instances of length  $k$ . Each of these algorithms corresponds to one particular subinstance  $(X_j, \mathcal{S}_j)$  of SETCOVER. Since these subinstances are disjoint, for any algorithm  $\mathbf{A}_j$  that makes  $r_j$  errors while using  $b_j$  advice bits, there is an algorithm  $\mathbf{B}_j$  using  $b_j$  advice bits that makes the same number of errors for the string  $s_j$  of the given instance of  $(q, \ell, k)$ -MULTISGKH. Thus, in total, the number of errors  $\mathbf{A}$  makes is the same as the number of errors made by some algorithm  $\mathbf{B}$  for the whole  $(q, \ell, k)$ -MULTISGKH instance. Due to Lemma 5, there is an algorithm  $\mathbf{C}$  that makes  $r$  errors while reading  $b$  advice bits for any instance of  $q$ -SGKH.

The competitive ratio achieved by  $\mathbf{A}$  is thus  $c = (\ell + (1 - \alpha) \cdot \ell k) / \ell = 1 + (1 - \alpha) \cdot k$ , hence  $\alpha = 1 - (c - 1)/k$ , where  $\alpha$  denotes the fraction of correct answers. Since we assume  $c \leq 1 + k(1 - 1/q)$ , which implies  $1/q \leq \alpha$ , we can directly apply Theorem 2 yielding that at least  $(1 - H_q(\frac{c-1}{k})) \cdot \ell k \cdot \log_2 q$  advice bits are necessary to be  $c$ -competitive. To measure in  $|\mathcal{S}| = m$  and in  $|X| = n$ , we calculate  $\ell = m/q^k$ , and  $\ell = (q - 1) \cdot n / (q^{k+1} - 1)$ , and the claim follows.  $\square$

Note that, in [17], a lower bound on achieving a constant competitive ratio was shown that is merely logarithmic in  $m$ . Thus, Theorem 5 yields an exponential improvement over the best previously known result.

## 4 The Online Maximum Clique Problem

In this section, we analyze the *online maximum clique* problem [10]. Here, in every time step, a vertex of a graph is given together with all edges to vertices that were already revealed in previous steps, and an online algorithm  $\mathbf{A}$  has to decide whether the newly revealed vertex becomes part of the solution.

For giving a reasonable cost function, we briefly give some considerations. First, assume the input graph  $G$  has a maximum clique of size  $n$  and the algorithm finds a clique of size  $n - 1$ . Then, intuitively, the cost of the solution should be  $n - 1$ , irrespective of how many vertices of the found clique are also part of a largest clique in  $G$ . On the other hand, assume the algorithm selects a vertex that is not connected to any vertex revealed afterwards. Unless this is the only vertex  $\mathbf{A}$  takes,  $\mathbf{A}$  does not output a clique (i. e., its solution is not feasible). To avoid this, it should be allowed to give an output where not all selected vertices are part of a clique, different to the situation in [10]. Even if  $\mathbf{A}$  gives an output in which many vertices form a large or even a maximum clique, but one additional vertex is selected, the output is not a clique, but very close to a relatively good or even optimal solution. Thus, this solution should have almost optimal cost. Then again, we should clearly prevent the algorithm from simply selecting all vertices that are given. For an output  $\mathbf{A}(I)$ , we therefore consider not only its size, but also the maximum clique  $C_{\mathbf{A}(I)}$  in the graph  $G_{\mathbf{A}(I)}$  restricted to the selected vertices  $\mathbf{A}(I)$ . Then, the solution becomes better the larger the maximum clique in  $G_{\mathbf{A}(I)}$  is, and it becomes worse as more vertices are selected that are not part of  $C_{\mathbf{A}(I)}$ . All in all, we propose the cost function given in the following definition.

**Definition 6 (Online Maximum Clique Problem).** *The online maximum clique problem (MAXCLIQUE) is the following online problem. The input is a graph  $G = (V, E)$  and the goal is to find a clique  $C \subseteq V$  in  $G$  of maximum size. In each time step  $i$ , one vertex  $v_i \in V$  is revealed together with all edges  $\{\{v_i, v_j\} \in E \mid j < i\}$ , and an online algorithm  $\mathbf{A}$  has to decide whether  $v_i \in C$  or not. Let  $\mathbf{A}(I)$  be the set of vertices selected by  $\mathbf{A}$ , and let  $C_{\mathbf{A}(I)}$  be a maximum clique in the graph  $G_{\mathbf{A}(I)}$ . The cost function is defined by  $\text{cost}(\mathbf{A}(I)) = |C_{\mathbf{A}(I)}|^2 / |\mathbf{A}(I)|$ .*

Clearly, for the optimal solution  $\text{Opt}(I)$  of a graph with a maximum clique  $C_{\text{opt}}$ , we have  $\text{cost}(\text{Opt}(I)) = \frac{|C_{\text{opt}}|}{|\text{Opt}(I)|} \cdot |C_{\text{opt}}| = |C_{\text{opt}}|$ , thus, the competitive ratio of  $\mathbf{A}$  on  $I$  is  $c = \frac{\text{cost}(\text{Opt}(I))}{\text{cost}(\mathbf{A}(I))} = \frac{|\mathbf{A}(I)|}{|C_{\mathbf{A}(I)}|} \cdot \frac{|C_{\text{opt}}|}{|C_{\mathbf{A}(I)}|}$ . In other words, the quality of the output of  $\mathbf{A}$  is given by the product of the two ratios  $|\mathbf{A}(I)|/|C_{\mathbf{A}(I)}|$  and  $|C_{\text{opt}}|/|C_{\mathbf{A}(I)}|$ . The first ratio measures how many useless vertices  $\mathbf{A}$  has taken and the second ratio measures how many correct vertices  $\mathbf{A}$  has not taken.

In order to give a lower bound on the advice complexity of MAXCLIQUE, we use our results for 2-SGKH. To this end, we investigate the following subclass of instances, where every instance corresponds to a particular bit string. Let  $s = s_1s_2\dots s_{n'}$  be a bit string of length  $n'$ , for some  $n' \in \mathbb{N}$ . We construct an input instance  $I_s$  for MAXCLIQUE corresponding to  $s$  as follows. Consider the graph  $G_{I_s} = (V(I_s), E(I_s))$  with  $n = 2n' + 2$  vertices. Let  $V(I_s) =$

$\{v_{(1,0)}, v_{(1,1)}, \dots, v_{(n'+1,0)}, v_{(n'+1,1)}\}$ , and let  $V'(I_s) = \{v_{(i,s_i)} \mid 1 \leq i \leq n'\}$  be the set of the  $n'$  vertices that correspond to the string  $s$ . Moreover, let

$$\begin{aligned} E(I_s) = & \{\{v_{(i,s_i)}, v_{(j,k)}\} \mid 1 \leq i < j \leq n', k \in \{0, 1\}\} \\ & \cup \{\{v, v_{(n'+1,0)}\}, \{v, v_{(n'+1,1)}\} \mid v \in V'(I_s)\} \cup \{\{v_{(n'+1,0)}, v_{(n'+1,1)}\}\}. \end{aligned}$$

Clearly, the vertices from  $V'(I_s)$  plus the vertices  $v_{(n'+1,0)}$  and  $v_{(n'+1,1)}$  form a unique optimal solution for  $I_s$  of size  $n' + 2$ . Although the vertices  $v_{(i,0)}$  and  $v_{(i,1)}$  are revealed separately (and after each vertex, the algorithm has to respond immediately), for the analysis, we combine them into one pair. After the first pair is revealed, the vertices of the second pair  $(v_{(2,0)}, v_{(2,1)})$  are given, and so on. An example for the string  $s = 00101$  of length 5 is given in Fig. 2. Assume that A knows that one vertex of each pair is part of the optimal solution. Then, we can see MAXCLIQUE as guessing one vertex per pair. Similar to guessing a string  $s$ , the correct decision can only depend on the input known so far, the history, and the given advice. However, in general, an algorithm has four options for every pair that is revealed, which are to take only the first vertex, only the second one, both, or none. As a next step, we show that, for any online algorithm with advice, it is the best strategy to take both vertices of any pair for which no advice is used. In this way, we derive an upper bound on the cost of any online algorithm for MAXCLIQUE that uses at most  $b$  advice bits.

**Lemma 6.** *Let B denote a best online algorithm with advice for 2-SGKH that reads  $b$  advice bits, and let  $s$  be any 2-SGKH instance of length  $n'$ . Let the number of bits of  $s$  that B guesses correctly be at most  $\alpha n'$ , where  $0 \leq \alpha \leq 1$ . Then, no online algorithm A for the corresponding MAXCLIQUE instance  $I_s$  reads  $b$  advice bits and achieves  $\text{cost}(A(I_s)) > (\alpha n' + (1 - \alpha)n' + 2)^2 / (\alpha n' + 2(1 - \alpha)n' + 2)$ . Furthermore, an online algorithm  $A^*$  that correctly guesses the same pairs as A and takes both vertices for all remaining pairs satisfies  $\text{cost}(A^*(I_s)) \geq \text{cost}(A(I_s))$ .*

In order to give a lower bound on the advice complexity, we analyze an online algorithm with advice that gets a sufficiently large number of advice bits to know  $\alpha n$  pairs and, following Lemma 6, takes both vertices for all unknown positions. Using our results from Section 2, we can prove the following theorem.

**Theorem 6.** *Any  $(c - \varepsilon)$ -competitive online algorithm A for MAXCLIQUE needs at least  $(1 - H_2(c - 1)) \frac{n-2}{2}$  advice bits, for any  $1 < c \leq 1.5$  and  $\varepsilon > 0$ .*

Note that, without advice, an online algorithm for MAXCLIQUE can achieve a competitive ratio of  $(2n' + 2)/(n' + 2) \approx 2$  by just taking every vertex.

**Acknowledgments.** The authors would like to thank Kfir Barhum and Richard Královič for enlightening discussions.

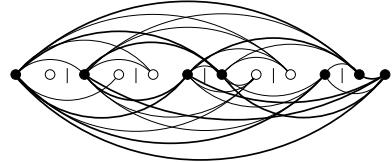


Fig. 2. The graph  $G_{00101}$

## References

1. Alon, N., Awerbuch, B., Azar, Y., Buchbinder, N., Naor, J.: The online set cover problem. *SIAM Journal on Computing* 39(2), 361–370 (2009)
2. Bianchi, M.P., Böckenhauer, H.-J., Hromkovič, J., Keller, L.: Online coloring of bipartite graphs with and without advice. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) *COCOON 2012*. LNCS, vol. 7434, pp. 519–530. Springer, Heidelberg (2012)
3. Borodin, A., El-Yaniv, R.: *Online Computation and Competitive Analysis*. Cambridge University Press (1998)
4. Böckenhauer, H.-J., Komm, D., Královic, R., Královic, R.: On the advice complexity of the  $k$ -server problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) *ICALP 2011*, Part I. LNCS, vol. 6755, pp. 207–218. Springer, Heidelberg (2011)
5. Böckenhauer, H.-J., Komm, D., Královic, R., Královic, R., Mömke, T.: On the advice complexity of online problems. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) *ISAAC 2009*. LNCS, vol. 5878, pp. 331–340. Springer, Heidelberg (2009)
6. Böckenhauer, H.-J., Komm, D., Královic, R., Rossmanith, P.: On the advice complexity of the knapsack problem. In: Fernández-Baca, D. (ed.) *LATIN 2012*. LNCS, vol. 7256, pp. 61–72. Springer, Heidelberg (2012)
7. Boyar, J., Kamali, S., Larsen, K.S., López-Ortiz, A.: Online bin packing with advice. Technical Report 1212.4016, arXiv 2012 (2012), <http://arxiv.org/abs/1212.4016>
8. Cohnen, G., Honkala, I., Litsyn, S., Lobstein, A.: *Covering Codes*. Elsevier (1997)
9. Dorrigiv, R., He, M., Zeh, N.: On the advice complexity of buffer management. In: Chao, K.-M., Hsu, T.-S., Lee, D.-T. (eds.) *ISAAC 2012*. LNCS, vol. 7676, pp. 136–145. Springer, Heidelberg (2012)
10. Demange, M., Paradon, X., Paschos, V.T.: On-line maximum-order induced hereditary subgraph problems. In: Jeffery, K., Hlaváč, V., Wiedermann, J. (eds.) *SOFSEM 2000*. LNCS, vol. 1963, pp. 327–335. Springer, Heidelberg (2000)
11. Dobrev, S., Královic, R., Pardubská, D.: How much information about the future is needed? In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) *SOFSEM 2008*. LNCS, vol. 4910, pp. 247–258. Springer, Heidelberg (2008)
12. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. *Theoretical Computer Science* 412(24), 2642–2656 (2011)
13. Forišek, M., Keller, L., Steinová, M.: Advice complexity of online coloring for paths. In: Dzediu, A.-H., Martín-Vide, C. (eds.) *LATA 2012*. LNCS, vol. 7183, pp. 228–239. Springer, Heidelberg (2012)
14. Guruswami, V., Rudra, A., Sudan, M.: *Essential Coding Theory* (2012), Draft available at <http://www.cse.buffalo.edu/~atri/courses/coding-theory/book/>
15. Hromkovič, J., Královic, R., Královic, R.: Information complexity of online problems. In: Hliněný, P., Kučera, A. (eds.) *MFCS 2010*. LNCS, vol. 6281, pp. 24–36. Springer, Heidelberg (2010)
16. Komm, D., Královic, R.: Advice complexity and barely random algorithms. *RAIRO ITA* 45(2), 249–267 (2011)
17. Komm, D., Královic, R., Mömke, T.: On the advice complexity of the set cover problem. In: Hirsch, E.A., Karhumäki, J., Lepistö, A., Prilutskii, M. (eds.) *CSR 2012*. LNCS, vol. 7353, pp. 241–252. Springer, Heidelberg (2012)
18. MacWilliams, F.J., Sloane, N.J.A.: *The Theory of Error-Correcting Codes*, 2nd edn. North-Holland Publishing Company (1978)

19. Moser, R., Scheder, D.: A full derandomization of Schöning's  $k$ -SAT algorithm. In: Proc. of STOC 2011, pp. 245–252. ACM (2011)
20. Renault, M.P., Rosén, A.: On online algorithms with advice for the  $k$ -server problem. In: Solis-Oba, R., Persiano, G. (eds.) WAOA 2011. LNCS, vol. 7164, pp. 198–210. Springer, Heidelberg (2012)
21. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communications of the ACM 28(2), 202–208 (1985)

# Online Algorithms for 1-Space Bounded 2-Dimensional Bin Packing and Square Packing

Yong Zhang<sup>1,2,\*</sup>, Francis Y.L. Chin<sup>2,\*\*</sup>, Hing-Fung Ting<sup>2,\*\*\*</sup>, Xin Han<sup>3,†</sup>,  
Chung Keung Poon<sup>4</sup>, Yung H. Tsin<sup>5,‡</sup>, and Deshi Ye<sup>6,§</sup>

<sup>1</sup> College of Mathematics and Computer Science, Hebei University, China

<sup>2</sup> Department of Computer Science, The University of Hong Kong, Hong Kong  
[{yzhang,chin,hfting}@cs.hku.hk](mailto:{yzhang,chin,hfting}@cs.hku.hk)

<sup>3</sup> School of Software, Dalian University of Technology, China  
[hanxin.mail@gmail.com](mailto:hanxin.mail@gmail.com)

<sup>4</sup> Department of Computer Science, City University of Hong Kong, Hong Kong  
[csckpoon@cityu.edu.hk](mailto:csckpoon@cityu.edu.hk)

<sup>5</sup> School of Computer Science, University of Windsor, Canada  
[peter@uwindsor.ca](mailto:peter@uwindsor.ca)

<sup>6</sup> College of Computer Science, Zhejiang University, China  
[yedeshi@zju.edu.cn](mailto:yedeshi@zju.edu.cn)

**Abstract.** In this paper, we study 1-space bounded 2-dimensional bin packing and square packing. A sequence of rectangular items (square items) arrive one by one, each item must be packed into a square bin of unit size on its arrival without any information about future items. When packing items, 90°-rotation is allowed. 1-space bounded means there is only one “active” bin. If the “active” bin cannot accommodate the coming item, it will be closed and a new bin will be opened. The objective is to minimize the total number of bins used for packing all items in the sequence.

Our contributions are as follows: For 1-space bounded 2-dimensional bin packing, we propose an online packing strategy with competitive ratio 5.06. A lower bound of 3.17 on the competitive ratio is proven. Moreover, we study 1-space bounded square packing, where each item is a square with side length no more than 1. A 4.3-competitive algorithm is achieved, and a lower bound of 2.94 on the competitive ratio is given. All these bounds surpass the previously best known results.

## 1 Introduction

Bin packing is one of the most fundamental problems in computer science. In the online fashion of bin packing, a sequence of items arrive one by one, each

---

\* Research supported by NSFC 11171086.

\*\* Research supported by HK RGC grant HKU-711709E.

\*\*\* Research supported by HK RGC grant HKU-716412E.

† partially supported by “the Fundamental Research Funds for the Central Universities(DUT12LK09)” and NSFC 11101065.

‡ Research supported by NSERC under grant NSERC 7811-2009.

§ Research supported by NSFC 11071215.

item must be packed into a bin on its arrival without any information about future items. The objective is minimizing the number of used bins for packing all items in the sequence.

Roughly speaking, the problem of online bin packing has two models: the *unbounded space* model and the *bounded space* model. In the unbounded space model, any bin can be used to pack the coming item if its empty space is large enough. In the bounded space bin packing, only “active” bins can be used to pack items, and the number of “active” bins is bounded by some constant. If all “active” bins cannot accommodate the coming item, one of such bins will be closed and a new bin will be opened to pack the coming item.

Our focus in this paper is the bounded space 2-dimensional bin packing, and the number of “active” bins is restricted to be one. We call it *1-space bounded 2-dimensional bin packing*. In this variant, the coming item is packed either in the “active” bin, or in a new “active” bin after the closure of the previous “active” bin. The closed bin cannot be used to pack items any more. In 1-space bounded bin packing,  $90^\circ$  rotation on item is allowed, otherwise, the performance ratio will be unbounded [5]. We also consider 1-space bounded square packing, where each item is a square with side length no more than 1. Our target is to find packing strategies for 1-space bounded 2-dimensional bin packing and square packing to minimize the number of used bins.

To measure the performance of the 1-space bounded 2-dimensional bin packing, we use the asymptotic competitive analysis, which is often used for online problems. For a sequence  $\sigma$  of items, let  $A(\sigma)$  and  $OPT(\sigma)$  denote the number of used bins by the online packing strategy  $A$  and the offline optimal algorithm  $OPT$ , respectively. The *asymptotic competitive ratio* of the online algorithm  $A$  is defined to be

$$R_A^\infty = \lim_{k \rightarrow \infty} \sup_{\sigma} \left\{ \frac{A(\sigma)}{OPT(\sigma)} \mid OPT(\sigma) = k \right\}.$$

## Related Works

Online bin packing has been studied for more than thirty years. For one-dimensional online bin packing, Johnson et al. [8] showed that the First Fit algorithm (FF) has an asymptotic competitive ratio of 1.7. Yao [13] improved the algorithm to obtain a better upper bound of  $5/3$ . Lee et al. [9] introduced the class of Harmonic algorithms, and showed that an asymptotic competitive ratio of 1.63597 is achievable. The best known upper bound is 1.58889, which was given by Seiden [10]. As for the lower bound of the competitive ratio of one dimensional bin packing, Yao [13] showed that no online algorithm can have an asymptotic competitive ratio less than 1.5. The best known lower bound is 1.54014 [12]. For two-dimensional online bin packing, Seiden and van Stee [11] showed an upper bound of 2.66013 by implementing the Super Harmonic Algorithm. The best known upper bound of the competitive ratio for two dimensional bin packing is 2.5545, which was given by Han et al. [6]. The best known lower bound is 1.907 [1].

For bounded space bin packing, Harmonic algorithm by Lee et al. [9] can be applied to one dimensional case, the competitive ratio is 1.69103 when the

number of active bins goes to infinity. Csirik and Johnson [2] presented an 1.7-competitive algorithm ( $K$ -Bounded Best Fit algorithms ( $BBF_K$ )) for one dimensional bin packing using  $K$  active bins, where  $K \geq 2$ . For multi-dimensional case, Epstein et al. [4] gave a  $1.69103^d$ -competitive algorithm using  $(2M - 1)^d$  active bins, where  $M \geq 10$  is an integer such that  $M \geq 1/(1 - (1 - \varepsilon)^{1/(d+2)}) - 1$ ,  $\varepsilon > 0$  and  $d$  is the dimension of the bin packing problem. For *1-space bounded 2-dimensional bin packing*, Fujita [5] first gave an  $O((\log \log m)^2)$ -competitive algorithm, where  $m$  is the width of the square bin and the size of each item is  $a \times b$  ( $a, b$  are integers and  $a, b \leq m$ ). Chin et al. proposed an 8.84-competitive packing strategy [3], and the upper bound was further improved to be 5.155 [14], they also gave the lower bound 3 for 1-space bounded two dimensional bin packing. If items are restricted to be squares, Zhang et al. [14] showed that the upper bound and lower bound of the competitive ratio are 4.5 and  $8/3$ , respectively. For 1-space bounded  $d$ -dimensional bin packing ( $d$  can be any integer), a  $4^d$ -competitive packing strategy was given in [15].

In the remaining part, 1-space bounded 2-dimensional bin packing is studied in Section 2, both upper and lower bounds of the competitive ratio are given; and in Section 3, we consider 1-space bounded square packing by showing the upper and lower bounds.

## 2 1-Space Bounded 2-Dimensional Bin Packing

### 2.1 Upper Bound

In this section, we give a 5.06-competitive algorithm for 1-space bounded 2-dimensional bin packing, improving the previous upper bound 5.15. Since  $90^\circ$ -rotation is allowed, we may assume that for each rectangular item  $(w, h)$ , the width is no less than the height, i.e.,  $w \geq h$ . The idea for packing rectangular items is a refinement of the technique used in [14].

The rectangular items are classified into three classes  $A$ ,  $B$  and  $C$  according to their widths:

$$\begin{aligned} A &= \{(w, h) | w > 1/2\}, \\ B &= \{(w, h) | 1/8 < w \leq 1/2\}, \text{ and} \\ C &= \{(w, h) | w \leq 1/8\}. \end{aligned}$$

For simplicity, let  $A$ -item denote an item belonging to class  $A$ .  $B$ -item and  $C$ -item are defined similarly.

Since all items are rectangular, in the packing strategy, items are packed with sides either vertical or parallel to the boundary of the bin. For  $A$ -items, the packing strategy pack them in the active bin using a top-down approach starting from the upper boundary of the bin. Since the width of each  $A$ -item is strictly larger than  $1/2$ , any two  $A$ -items cannot share the same horizontal line within a bin. The width of  $B$ -item and  $C$ -item are upper bounded by  $1/2$ , in the packing strategy, they are packed either in the left half side or in the right half side of the bin by using a bottom-up approach starting from the lower boundary of the bin. The heights of the left side and the right side are packed as balance as possible.

If an item cannot be packed into the bin using the strategy, the active bin will be closed and a new one will be opened to pack this item.

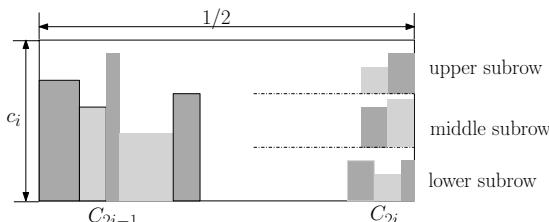
Let the *occupation ratio* to be the utilization of some area in the bin. For an *A*-item  $(w_1, h_1)$ , since no other items share the same horizontal line in the bin, the strip  $(1, h_1)$  containing this *A*-item cannot be used to pack other item, thus, the *occupation ratio* of this strip is  $\frac{w_1 \cdot h_1}{1 \cdot h_1}$ . Since the width of each *A*-item is larger than  $1/2$ , the occupation ratio of any strip containing *A*-item is strictly larger than  $1/2$ . For an *B*-item  $(w_2, h_2)$ , since it is packed into either the left side or the right side, the strip  $(1/2, h_2)$  containing this *B*-item cannot be used to pack other item, thus, the occupation ratio of this strip is  $\frac{w_2 \cdot h_2}{1/2 \cdot h_2}$ . Since the width of *B*-item is in between  $1/8$  and  $1/2$ , the occupation ratio of any strip containing *B*-item is at least  $1/4$ . Note that *C*-item may be very tiny, if it is packed by using the same way as for *A*-item or *B*-item, the wastage will be very large and the performance will be bad.

We first consider how to pack *C*-item. The width of *C*-item is upper bounded by  $1/8$ , but it is not lower bounded by any positive value. *C*-items are classified into subclasses  $C_1, C_2, C_3, \dots$  according to their widths. Let  $c_1 = 1/8$ ,  $c_i = a \cdot c_{i-1}$  for  $i > 1$ , where  $a = (6 - \sqrt{21})/15 < 1/9$ . We say

$$\text{an item } (w, h) \text{ belongs to subclass } \begin{cases} C_{2i-1} & \text{if } 3a \cdot c_i < w \leq c_i \\ C_{2i} & \text{if } a \cdot c_i < w \leq 3a \cdot c_i \end{cases}$$

Thus,  $c_i$  is the maximal width of items from subclass  $C_{2i-1}$  and  $3a \cdot c_i$  is the maximal width of items from subclass  $C_{2i}$ . Each item belonging to subclass  $C_{2i-1}$  or  $C_{2i}$  ( $i > 0$ ) can be packed into a row with height  $c_i$  and width  $1/2$ . The items from subclasses  $C_{2i-1}$  are packed from left to right while the items from subclass  $C_{2i}$  are packed from right to left in three subrows (upper, middle and lower), keeping the lengths of these three subrows balanced at all times (that means a new item is always packed into the subrow with the shortest packed length). Note that *C*-item is packed with a  $90^\circ$ -rotation. Figure 1 depicts a row with packed items from subclass  $C_{2i-1}$  and  $C_{2i}$ . When handling an item from subclass  $C_{2i-1}$  (or  $C_{2i}$ ), a new row of height  $c_i$  will be created if the existing rows with height  $c_i$  cannot accommodate this item by the above packing method.

Consider the packing of *C*-item. If there is more than one row for the subclasses  $C_{2i-1}$  and  $C_{2i}$ , the last row could be almost empty and the non-last rows are almost full. The total height of the last rows is at most



**Fig. 1.** Packing  $C_{2i-1}$  (or  $C_{2i}$ )-items ( $i > 0$ ) into a row

$$\sum_{i>0} c_i = \frac{c_1}{1-a} = \frac{1/8}{1-(6-\sqrt{21})/15} \approx 0.138.$$

Now we analyze the occupation ratio of the non-last rows for subclasses  $C_{2i-1}$  and  $C_{2i}$ . In the left side, suppose the occupied length is  $x$ , thus, the total occupation in the left side is at least

$$3a \cdot c_i \cdot x.$$

In the right side, suppose the length of the longest occupied subrow is  $y_1$  and the length of the shortest occupied subrow is  $y_2$ , the total occupation in the right side is at least

$$3a \cdot c_i \cdot y_2 + a \cdot c_i \cdot (y_1 - y_2).$$

Since  $y_1 - y_2 \leq 3a \cdot c_i$  (that is owing to the balanced packing in the right side), the above formula is at least

$$3a \cdot c_i \cdot y_1 - 6a^2 \cdot c_i^2.$$

In the left side, the height of the item may be larger than  $3a \cdot c_i$ . Using an amortized analysis as follows, if packing an item  $(w, h)$  which belongs to subclass  $C_{2i-1}$  will create a new row, this item contributes  $\max\{0, h^2 - 3a \cdot c_i \cdot h\}$  to the row which cannot pack it. If  $h > 3a \cdot c_i$ , the contribution of this item in the newly created row is  $3a \cdot c_i \cdot h$  and the remaining area of this item is larger than  $h^2 - 3a \cdot c_i \cdot h$ , which will contribute to the previous row.

**Lemma 1.** *For any non-last row with height  $c_i$  ( $i > 0$ ), the amortized occupation ratio is at least  $1/4$ .*

The packing strategy can be described as follows.

---

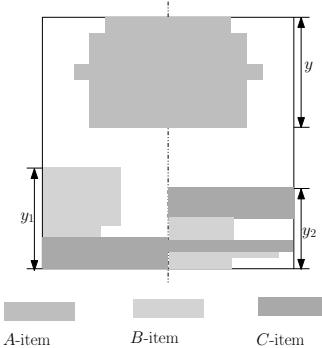
**Algorithm Packing-Bin:** for 1-space bounded 2-dimensional bin packing

---

- 1:  $A$ -items are packed in a top-down order starting from the top boundary of the bin.
  - 2:  $B$ -items and  $C$ -items are packed in a bottom-up order along both the left and right side of the square bin, keeping the heights of these two sides balanced at all times, i.e., a new  $B$ -item or newly created row of  $C$ -item is always packed on the side with smaller height.
  - 3: If there is insufficient space to pack a new item ( $A$ -item,  $B$ -item) or create a new row for the coming  $C$ -item, the bin is closed and a new bin is opened to pack the new item or row.
- 

An example of a packing configuration by applying the algorithm Packing-Bin is illustrated in Figure 2. In this configuration, the height of the packed  $A$ -items is  $y$ , the left and right sides of the packed  $B$ -items and  $C$ -items are of height  $y_1$  and  $y_2$  respectively. In this example,  $y_1 > y_2$ , according to the algorithm, if a  $B$ -item comes, we pack it in the right side.

For a given sequence of items, suppose the number of bins used by the packing strategy Packing-Bin is  $n$ . Let  $o_A^i$ ,  $o_B^i$  and  $o_C^i$  be the occupied space of  $A$ -,  $B$ - and  $C$ - items in the  $i$ -th bin respectively. The average occupation for all the bins is  $\sum_{i=1}^n (o_A^i + o_B^i + o_C^i)/n$ .



**Fig. 2.** Packing rectangular items into a square bin

Consider the packing configuration of the  $i$ -th bin as shown in Figure 2, assume that the height of the packed  $A$ -items is  $y$ , the left and right sides of the packed  $B$ -items and  $C$ -items are of height  $y_1$  and  $y_2$  respectively. W.l.o.g.,  $y_1 \geq y_2$ . We have

$$\begin{aligned} o_A^i &\geq y/2 \\ o_B^i &\geq (y_1 + y_2 - \sum_{j \geq 1} c_j - m)/8 \geq (y_1 + y_2 - 0.138 - m)/8 \\ o_C^i &\geq m/8 \end{aligned}$$

where  $m$  is the total height of the non-last rows of  $C$ -items.

By using an amortized analysis, if an  $A$ -item or an  $B$ -item cannot be packed into the active bin by the packing algorithm, this item will contribute some area to the just recently closed bin. For the  $i$ -th bin, let  $q_A^i$  and  $q_B^i$  be the contribution of  $A$ -items and  $B$ -items to the  $(i-1)$ -th bin, and  $p_A^i$  and  $p_B^i$  be the remaining areas of  $A$ -items and  $B$ -items in the  $i$ -th bin. Formally, if an  $A$ -item  $(w, h)$  cannot be packed into the  $i$ -th bin,

$$q_A^i = \frac{w \cdot h}{2};$$

if an  $B$ -item  $(w, h)$  cannot be packed into the  $i$ -th bin,

$$q_B^i = \begin{cases} w \cdot h/2 & \text{if } 1/4 \leq h < 1/2 \\ h^2 - h/8 & \text{if } 1/8 \leq h < 1/4 \end{cases}$$

For this item, the remaining area is

$$\begin{cases} w \cdot h/2 \geq h/8 & \text{if } 1/4 \leq h < 1/2 \\ w \cdot h - h^2 + h/8 \geq h/8 & \text{if } 1/8 \leq h < 1/4 \end{cases}$$

Since we focus on the asymptotic performance, when  $n$  is very large, we have

$$\frac{\sum_{i=1}^n (o_A^i + o_B^i + o_C^i)}{n} \geq \min_{1 \leq i \leq n} \{p_A^i + p_B^i + o_C^i + q_A^{i+1} + q_B^{i+1}\} \quad (1)$$

By considering cases on the first item which cannot be packed in the  $i$ -th bin, we have the following conclusion.

**Theorem 1.** *The competitive ratio of the packing strategy **Packing-Bin** is at most 5.06.*

## 2.2 Lower Bound

In this part, we prove that the lower bound of the competitive ratio is at least 3.167, improving the previous bound 3.

**Theorem 2.** *The lower bound of the competitive ratio for 1-space bounded 2-dimensional bin packing is at least 3.167.*

*Proof.* Consider a sequence of items:  $S = \{X_1, X_2, \dots, X_{2n}, A_1, A_2, B_2, \dots, A_n, B_n, T_1, T_2, \dots, T_n\}$ .

In the first part of the item sequence containing all the  $X_i$  items,

$$\begin{aligned} X_{2i-1} &= (1/2 + i \cdot \epsilon, 1/2 + i \cdot \epsilon) \\ X_{2i} &= (1/2 - (i-1) \cdot \epsilon, 1/2 - (i-1) \cdot \epsilon) \end{aligned}$$

in which  $\epsilon = o(1/n^2)$ . It can be verified that no online algorithm can pack any two consecutive items into one unit square bin because the sum of the edge lengths of any two consecutive  $X$ -items is larger than 1. Thus, at least  $2n$  bins are used for packing all these  $X_i$  items. However  $X_{2i-1}$  and  $X_{2i+2}$  can be packed into the same bin in the optimal packing.

In the second part of the item sequence containing all  $A_i$  and  $B_i$  items,

$$\begin{aligned} A_i &= (1/3 + \epsilon, 2/3 + \epsilon_{i+1}) \\ B_i &= (1/3 + \epsilon, 1/3 - \epsilon_i) \end{aligned}$$

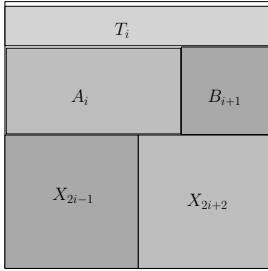
in which  $\epsilon = o(1/n^2)$ ,  $\epsilon_i < \epsilon_{i+1}$  for  $i \geq 1$ , and  $0 < \epsilon_i = o(1/n^2)$  for all  $i \geq 1$ . In the online fashion,  $A_i$  and  $A_{i+1}$  cannot be packed into the same bin. Thus, no online algorithm can pack the second part of the item sequence by using less than  $n$  bins. However  $X_{2i-1}$ ,  $X_{2i+2}$ ,  $A_i$ ,  $B_{i+1}$  can be packed into one bin in the optimal packing.

In the third part of the item sequence containing all  $T_i$  items,

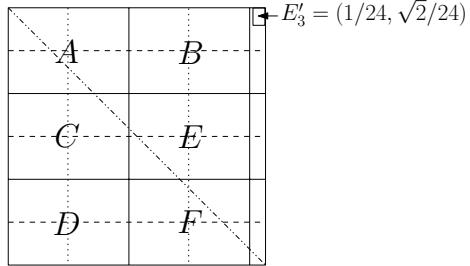
$$T_i = (1, 1/7 + \epsilon)$$

in which  $\epsilon = o(1/n^2)$ . It can be verified that any bin can contain at most 6 items from this part. However, one  $T_i$  can be packed together with  $X_{2i-1}$ ,  $X_{2i+2}$ ,  $A_i$ ,  $B_{i+1}$  in the optimal packing, as shown in Figure. 3.

Combine these three parts, note that our focus is the asymptotic performance, there is no online algorithm which can pack all items in this sequence within  $2n+n+n/6$  bins, while the optimal strategy only uses  $n$  bins. Thus, we conclude that no online algorithm can achieve a competitive ratio less than 3.167 for 1-space bounded 2-dimensional bin packing.  $\square$



**Fig. 3.** The optimal packing



**Fig. 4.** Partition of unit bin

### 3 1-Space Bounded Square Packing

In 1-space bounded square packing, each item is a square with side length no more than 1. In our packing strategy, square items are packed in bricks where a *brick* is a rectangle with aspect ratio  $\sqrt{2}$ . Packing square items in bricks is a popular method. The interesting property of this method is: a brick can be partitioned into two smaller congruent bricks of the same size. Thus, packing a square into a brick can be done recursively. Given a square  $Q$ , let  $S(Q)$  denote the smallest brick which can contain  $Q$ . Let  $|R|$  denote the area of rectangle  $R$ .

The following is a modified algorithm similar to [7] for packing a square  $Q$  in a brick  $T$ .

---

#### Algorithm Brick( $Q, T$ ): Packing a square $Q$ in brick $T$

---

- 1: If there is no empty brick in  $T$  of size greater than or equal to  $S(Q)$ , then give up packing  $Q$  in  $T$ .
  - 2: Else pack  $Q$  in  $T$  as follows:
    - if there is an empty brick congruent to  $S(Q)$ , then pack  $Q$  into it;
    - else partition the smallest empty brick  $P$  that is larger than  $S(Q)$  into two congruent bricks  $P_1$  and  $P_2$ . Assume  $P_1$  is the left (or the upper) one. Recursively execute  $Brick(Q, P_1)$ .
- 

**Lemma 2.** [7] *If the above algorithm cannot pack an item  $Q$  in a brick  $B$ , then all empty bricks in  $B$  are smaller than  $S(Q)$ . Furthermore, there is at most one empty brick with area  $|S(Q)|/2^i$  for each  $i = 1, 2, \dots$ , and the total area of the empty bricks is less than  $|S(Q)|$ .*

**Lemma 3.** *If  $Q$  is packed in a brick congruent to  $S(Q)$ , then at least  $1/(2\sqrt{2})$  of this brick is occupied.*

### 3.1 Upper Bound

We partition each unit bin as shown in Figure 4. Bricks  $A$  to  $F$  are of the same size  $(1/3, \sqrt{2}/3)$ , and each brick can be further partitioned into two congruent bricks. We call an item *small*, *middle*, and *large* if the edge length  $\ell$  satisfies  $\ell \leq 1/3$ ,  $1/3 < \ell \leq 1/2$ , and  $\ell > 1/2$ , respectively. There is a small brick  $E'_3 = (1/24, \sqrt{2}/24)$  in the right-top of the bin. This brick is used only in some special cases, which will be described in later analysis. The packing strategy is described as follows.

---

**Algorithm Packing-Square:** For 1-space bounded square packing

---

- 1: For a small item  $s$ , by using the algorithm **Brick()**, we search  $A, B, C, D, E, F$ , in the listed order, for an  $S(s)$  to pack  $s$ . E.g., if **Brick( $s, A$ )** cannot pack  $s$ , then consider **Brick( $s, B$ )**.
  - 2: For a middle item  $s$ , we search the available position in the following order to pack item  $s$ .
    - 1) the left-bottom corner of the bin;
    - 2) the right-bottom corner of the bin;
    - 3) if there is a middle item  $s'$  packed on the right-bottom corner of the bin, consider the position immediately to the left of  $s'$ ;
    - 4) the right-top corner of the bin; and
    - 5) the left-top corner of the bin;

▷ Note that packing a middle item in the left-corner of the bin may overlap with bricks  $E$  and  $F$  since the side length may be larger than  $\sqrt{2}/3$ . In this case, bricks  $E$  and  $F$  are slightly shifted to the right such that there is no overlap with the packed middle item.
  - 3: For a large item  $s$ , we pack it at the right-bottom corner of the unit bin.
  - 4: If item  $s$  cannot be packed into the active bin by using the above rules, this bin will be closed then a new bin will be opened to pack  $s$ .
- 

Now we analyze the competitive ratio of the above algorithm. For a sequence of square items, assume the offline optimal packing strategy uses  $n$  bins, and our algorithm uses  $x + y + z$  bins, where  $x$  is the number of bins containing a large item,  $y$  is the number of bins not containing large item and closed by the packing of a large item, and  $z$  is the number of remaining bins. In the optimal packing, a bin contains at most one large item, thus,  $x \leq n$ . From the definition of  $y$ , for each bin with large item, there is at most one previous bin counted in those  $y$  bins, thus,  $y \leq x$ .

By using the idea of amortized analysis, if the amortized occupation of those  $x$ ,  $y$ , and  $z$  items are  $a$ ,  $b$ , and  $c$ , respectively, the total area of the sequence of items is at least  $ax + by + cz$ , which is upper bounded by  $n$  since the optimal packing uses  $n$  bins. Therefore, the competitive ratio is

$$\max (x + y + z)/n \quad (2)$$

$$\text{s.t. } ax + by + cz \leq n \quad \text{and}$$

$$y \leq x \leq n$$

In the amortized analysis, if there is a large item  $s$  with side length  $\ell > 1/2$  in a bin, this item contributes  $\ell^2 - 1/4$  to the previous bin and the remaining area which contributes to its packed bin is  $1/4$ ; if a bin is closed by the packing of a middle item  $s'$  with side length  $1/3 < \ell' \leq 1/2$ , this item contributes  $(\ell'^2 - 1/9)/2$  to the previous bin, and the remaining area which contributes to its packed bin is  $\ell'^2 - (\ell'^2 - 1/9)/2 = \ell'^2/2 + 1/18 \geq 1/9$ .

For those  $x$  bins containing large items, the remaining area is at least  $1/4$ . Thus, we may set  $a = 1/4$ .

For those  $y$  bins, they are closed by the packing of large items. The following lemma analyzed the amortized occupation in this part of bins.

**Lemma 4.** *The amortized occupation in those  $y$  bins is at least  $(10 - 6\sqrt{2})/9$ .*

Thus,  $b = (10 - 6\sqrt{2})/9 \approx 0.1683$ .

Now we give the amortized occupation for the remaining  $z$  bins.

**Lemma 5.** *The amortized occupation in those  $z$  bins is at least  $1/4$ .*

Thus,  $c = 1/4$ .

Next, we give the competitive ratio of the algorithm Packing-Square.

**Theorem 3.** *The competitive ratio of the algorithm Packing-Square is at most 4.3268.*

*Proof.* Substituting the values of  $a$ ,  $b$ , and  $c$  into Formula (2),

$$\begin{aligned} & \max \quad (x + y + z)/n \\ \text{s.t. } & x/4 + 0.1683y + z/4 \leq n \quad \text{and} \\ & y \leq x \leq n \end{aligned} \tag{3}$$

Therefore,

$$(x + y + z)/4 \leq n + 0.0817y \leq 1.0817n.$$

The competitive ratio of the packing strategy is at most

$$\frac{x + y + z}{n} \leq 1.0817 \cdot 4 = 4.3268$$

□

### 3.2 Lower Bound

Now we derive a lower bound of the competitive ratio for 1-space bounded square packing. Roughly speaking, the adversary sends items in phases.

- In the first phase, the side lengths of the coming items are very close to  $1/2$ .
- In the second phase, the side lengths of the items are very close to  $1/3$ .
- ...

The high level idea underlying the lower bound proof is as follows: The adversary constructs a sequence with  $2n$  items in the first phase,  $3n$  items in the second phase, ... such that no online packing algorithm can use less than  $2n$  bins for the first phase,  $3n/4$  bins for the second phase, ... But for the optimal packing strategy,  $n$  bins is sufficient to pack all items.

**Theorem 4.** *There is no online algorithm with a competitive ratio less than 2.75 for 1-space bounded square packing.*

*Proof.* As mentioned above, the adversary sends items in phases. In the first phase, the item sequence is  $(Y_1, X_1, Y_2, X_2, \dots, Y_n, X_n)$ . Let  $\epsilon = o(1/n^2)$ . Let  $y_i$  and  $x_i$  denote the side length of  $X_i$  and  $Y_i$ , respectively. The side lengths of these items are as follows.

$$y_i = 1/2 - (n + 1 - i)\epsilon$$

$$x_i = 1/2 + (n + 2 - i)\epsilon$$

It can be verified that any two adjacent items cannot be packed into one bin. Thus, no online algorithm can pack these items by using less than  $2n$  bins. However, items  $Y_i$  and  $X_{i+1}$  can be packed into the same bin by the optimal strategy.

In the second phase, the adversary sends  $3n$  items, the arrival order is  $(U_3, U_4, W_1, W_2, U_5, U_6, W_3, W_4, \dots, U_{n-1}, U_n, W_{n-3}, W_{n-2}, U_1, U_2, V_1, V_2, \dots, V_n, W_{n-1}, W_n)$ . Let  $u_i$ ,  $v_i$ , and  $w_i$  denote the side length of  $U_i$ ,  $V_i$ , and  $W_i$ , respectively. Let  $\epsilon \ll \epsilon_1$ ,  $\epsilon_i < \epsilon_{i+1}$  for  $i \geq 1$ ,  $\epsilon_{2i+1} > \epsilon_{2i-1} + \epsilon_{2i} + 2\epsilon$  for  $i \geq 1$ , and  $\epsilon_i = o(1/n^2)$  for any  $i$ . The side lengths of these items are as follows.

$$u_i = 1/3 + \epsilon_i$$

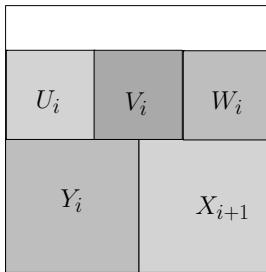
$$v_i = 1/3 + \epsilon$$

$$w_i = 1/3 - \epsilon - \epsilon_i$$

It can be verified that except  $(U_n, W_{n-3}, W_{n-2}, U_1, U_2, V_1)$  and  $(V_n, V_n, V_n, W_{n-1}, W_n)$ , any adjacent five items cannot be packed into the same bin. Since we consider the asymptotic performance, i.e.,  $n$  is very large, no online algorithm can pack these items by using  $3n/4$  bins. However, items  $U_i$ ,  $V_i$ , and  $W_i$  can be packed together with  $Y_i$  and  $X_{i+1}$ .

After the second phase, any online algorithm uses at least  $2n + 3n/4$  bins, while the optimal packing strategy only uses  $n$  bins. Thus, the competitive ratio is at least 2.75.  $\square$

For the above item sequence, the optimal packing in a bin is shown in Figure 5. There are still some free space in the upper part of the optimal packing. We can fully utilize these free space to force the online algorithm uses more bins. In the optimal packing, the height of the empty part is around  $1/6$ . The adversary may design another phase with  $7n$  items whose side lengths are around  $1/7$ , such that no consecutive 37 items can be packed into the same bin, thus,  $7n/36$  bins are needed for the online packing. In the optimal strategy, 7 item can be packed into the upper part of the optimal packing as shown in Figure 5. Thus, we have the following Theorem.



**Fig. 5.** The optimal packing

**Theorem 5.** *There is no online algorithm with a competitive ratio less than 2.94 for 1-space bounded square packing.*

## References

1. Blitz, D., van Vliet, A., Woeginger, G.J.: Lower bounds on the asymptotic worst-case ratio of on-line bin packing algorithms (1996) (unpublished manuscript)
2. Csirik, J., Johnson, D.S.: Bounded Space On-Line Bin Packing: Best is Better than First. *Algorithmica* 31, 115–138 (2001)
3. Chin, F.Y.L., Ting, H.-F., Zhang, Y.: 1-Bounded Space Algorithms for 2-Dimensional Bin Packing. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) *ISAAC 2009. LNCS*, vol. 5878, pp. 321–330. Springer, Heidelberg (2009)
4. Epstein, L., van Stee, R.: Optimal Online Algorithms for Multidimensional Packing Problems. *SIAM Journal on Computing* 35(2), 431–448 (2005)
5. Fujita, S.: On-Line Grid-Packing with a Single Active Grid. *Information Processing Letters* 85, 199–204 (2003)
6. Han, X., Chin, F.Y.L., Ting, H.F., Zhang, G., Zhang, Y.: A New Upper Bound on 2D Online Bin Packing. *ACM Transactions on Algorithms* 7(4), 50 (2011)
7. Januszewski, J., Lassak, M.: On-line packing sequences of cubes in the unit cube. *Geometriae Dedicata* 67, 285–293 (1997)
8. Johnson, D.S., Demers, A.J., Ullman, J.D., Garey, M.R., Graham, R.L.: Worst-Case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing* 3(4), 299–325 (1974)
9. Lee, C.C., Lee, D.T.: A simple on-line bin packing algorithm. *J. Assoc. Comput. Mach.* 32, 562–572 (1985)
10. Seiden, S.S.: On the online bin packing problem. *J. ACM* 49, 640–671 (2002)
11. Seiden, S., van Stee, R.: New bounds for multi-dimensional packing. In: Proc. of SODA, pp. 486–495 (2002)
12. van Vliet, A.: An improved lower bound for on-line bin packing algorithms. *Information Processing Letters* 43, 277–284 (1992)
13. Yao, A.C.-C.: New Algorithms for Bin Packing. *Journal of the ACM* 27, 207–227 (1980)
14. Zhang, Y., Chen, J., Chin, F.Y.L., Han, X., Ting, H.-F., Tsin, Y.H.: Improved Online Algorithms for 1-Space Bounded 2-Dimensional Bin Packing. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) *ISAAC 2010, Part II. LNCS*, vol. 6507, pp. 242–253. Springer, Heidelberg (2010)
15. Zhang, Y., Chin, F.Y.L., Ting, H.-F., Han, X., Chang, Z.: Online Algorithm for 1-Space Bounded Multi-dimensional Bin Packing. In: Atallah, M., Li, X.-Y., Zhu, B. (eds.) *FAW-AAIM 2011. LNCS*, vol. 6681, pp. 308–318. Springer, Heidelberg (2011)

# Improved Lower Bounds for the Online Bin Packing Problem with Cardinality Constraints

Hiroshi Fujiwara<sup>1,\*</sup> and Koji Kobayashi<sup>2</sup>

<sup>1</sup> Toyohashi University of Technology

h-fujiwara@cs.tut.ac.jp

<sup>2</sup> National Institute of Informatics

kobaya@nii.ac.jp

**Abstract.** The bin packing problem has been extensively studied and numerous variants have been considered. The *k-item bin packing* problem is one of the variants introduced by Krause et al. in Journal of the ACM 22(4). In addition to the formulation of the classical bin packing problem, this problem imposes a *cardinality constraint* that the number of items packed into each bin must be at most  $k$ . For the *online* setting of this problem, i.e., the items are given one by one, Babel et al. provided lower bounds  $\sqrt{2} \approx 1.41421$  and 1.5 on the asymptotic competitive ratio for  $k = 2$  and 3, respectively, in Discrete Applied Mathematics 143(1-3). For  $k \geq 4$ , some lower bounds (e.g., by van Vliet in Information Processing Letters 43(5)) for the online bin packing problem, i.e., a problem without cardinality constraints, can be applied to this problem.

In this paper we consider the online  $k$ -item bin packing problem. First, we improve the previous lower bound 1.41421 to 1.42764 for  $k = 2$ . Moreover, we propose a new method to derive lower bounds for general  $k$  and present improved bounds for various cases of  $k \geq 4$ . For example, we improve 1.33333 to 1.5 for  $k = 4$ , and 1.33333 to 1.47058 for  $k = 5$ .

## 1 Introduction

The bin packing problem is a classical problem in the field of computer science, which has been most extensively studied. This problem is defined as follows. We are given a sequence of *items*, each of which has a *size* in  $(0, 1]$ , as an input, and an infinite number of *bins*. Each item has to be packed into one of the bins, and the sum of sizes of items packed into each bin has to be at most one. A bin that contains at least one item is said to be *non-empty*. The goal of this problem is to minimize the number of non-empty bins.

The bin packing problem has been studied also in the *online* setting: The items are given one by one, and each item has to be packed before the next one is given. This problem is quite important in both theoretical and applied aspects, and much work has been done on this problem (e.g. [10,13,11,2]). Online algorithms are usually evaluated using competitive analysis [3,12]. For any sequence  $\sigma$  of

---

\* This work was supported by KAKENHI (23700014 and 23500014).

**Table 1.** Previous results and our results for the online  $k$ -item bin packing problem

| $k$       | lower bound                                                                                     | upper bound                                    |
|-----------|-------------------------------------------------------------------------------------------------|------------------------------------------------|
| 2         | $\sqrt{2} (\approx 1.41421)$ [1] $\rightarrow 1.42764$ [this paper]                             | $1 + \frac{\sqrt{5}}{5} (\approx 1.44721)$ [1] |
| 3         | 1.5 [1]                                                                                         | 1.75 [5]                                       |
| 4         | $\frac{4}{3} (\approx 1.33333)$ [13] $\rightarrow 1.5$ [this paper]                             | $\frac{71}{38} (\approx 1.86843)$ [5]          |
| 5         | $\frac{4}{3} (\approx 1.33333)$ [13] $\rightarrow \frac{25}{17} (\approx 1.47058)$ [this paper] | $\frac{771}{398} (\approx 1.93719)$ [5]        |
| 6         | 1.5 [14]                                                                                        | $\frac{287}{144} (\approx 1.99306)$ [5]        |
| 7 to 9    | 1.5 [14]                                                                                        |                                                |
| 10 to 41  | 1.5 [14] $\rightarrow$ (See Table 2 in Section 3.)                                              | 2 [1]                                          |
| 42 to 293 | $\frac{217}{141} (\approx 1.53900)$ [13]                                                        |                                                |
| $\infty$  | $\frac{248}{161} (\approx 1.54037)$ [2]                                                         | 1.58889 [11]                                   |

items, and any algorithm  $ALG$ , let  $C_{ALG}(\sigma)$  denote the number of  $ALG$ 's non-empty bins for  $\sigma$ . Let  $OPT$  be an optimal offline algorithm. Then, for any online algorithm  $ON$ , define  $R_{ON} = \limsup_{n \rightarrow \infty} \sup_{\sigma} \{C_{ON}(\sigma)/C_{OPT}(\sigma) \mid C_{OPT}(\sigma) = n\}$ , which we call the *asymptotic competitive ratio* (also known as the *asymptotic performance ratio*) of  $ON$ .

A constraint that the number of items packed into one bin is somehow restricted seems quite realistic in application. For example, there exists the minimum size of files used by a computer, and the number of files stored on the computer is thus bounded. In light of this situation, Krause et al. [8,9] introduced the  *$k$ -item bin packing* problem, in which the *cardinality constraint* that each bin can contain at most  $k$  items is imposed. (They defined this problem as a scheduling problem.) This problem has been well studied in both the offline and online settings.

*Previous Results and Our Results.* In the *online  $k$ -item bin packing* problem, in which items are given in an online manner and the number of items in a bin has to be at most  $k$ , Babel et al. [1] showed that for  $k = 2$ , the asymptotic competitive ratio of any online algorithm is at least  $\sqrt{2} \approx 1.41421$ . Also, they presented a lower bound of 1.5 when  $k = 3$  using the method by Yao [14]. Moreover, for larger  $k$ , various lower bounds by van Vliet [13], Yao [14], and Balogh et al. [2] for the online bin packing problem, i.e., a problem without cardinality constraints, can be applied to the online  $k$ -item bin packing problem. We mention that the lower bounds for  $k = 4$  and 5 are straightforwardly given by manipulating the method in [13]. (See Table 1.)

In this paper, we consider the online  $k$ -item bin packing problem. First, we show that the asymptotic competitive ratio of any algorithm is at least  $\bar{r} \approx 1.42764$  for  $k = 2$ , where  $\bar{r}$  is the root of the equation  $2r^3 - 17r^2 + 30r - 14 = 0$  between  $\frac{4}{3}$  and  $\frac{3}{2}$ , which improves the previous lower bound. Second, we extend the method to obtain lower bounds for the online bin packing problem by van Vliet [13] and get various improved lower bounds for various cases of  $k \geq 4$ . For

example, we improve 1.33333 to 1.5 for  $k = 4$ , and 1.33333 to 1.47058 for  $k = 5$ . (See Table 1, and Table 2 in Section 3.)

*Related Results.* In the online  $k$ -item bin packing problem, Krause et al. [8,9] showed that for any  $k$ , the asymptotic competitive ratio of the most basic algorithm FIRSTFIT is at most  $2.7 - 12/5k$ . Babel et al. [1] established an algorithm whose asymptotic competitive ratio is at most 2 for any  $k$ . Moreover, Babel et al. [1] and Epstein [5] designed algorithms for small  $k$ . These results are also presented in Table 1. In addition, Epstein [5] established a bounded space algorithm. She showed that its asymptotic competitive ratio is at most 2.69104, and is asymptotically optimal. Note that while a bounded space algorithm always has only a constant number of bins available to accept items, the other results described above, including our new results, focus on unbounded space algorithms. There are some studies [7,4,6] about approximation algorithms for the  $k$ -item bin packing problem. Needless to say, the online bin packing problem (without cardinality constraints) has been much studied, and the best upper and lower bounds are 1.58889 by Seiden [11] and  $248/161 \approx 1.54037$  by Balogh et al. [2], respectively.

## 2 A Lower Bound for $k = 2$

In this section we present a lower bound of 1.42764 for  $k = 2$ . We first define an adversary, which determines the size of the next item adaptively according to the behavior of an online algorithm. The strategy of the adversary is chosen from the three strategies whose pseudocodes will be later given as ROUTINE1, 2, and 3, respectively.

We begin by mentioning three subroutines called by ROUTINE1, 2, and 3. See their pseudocodes SUBROUTINE1, 2, and 3 below. Roughly speaking, each subroutine gives a sequence of items while changing the size within a specified range. The only difference between them is just the termination conditions. Let us see the details. Each subroutine is called with four parameters: an online algorithm  $ON$  and three values  $\text{Min}$ ,  $\text{Max}$ , and  $\text{Length}$  with  $\text{Min} < \text{Max}$ . Each subroutine returns the current value of the internal variable  $\text{tmpMin}$ . The sizes of given items lie in  $(\text{Min}, \text{Max})$ . For ease of presentation, if an algorithm  $ALG$  is about to put an item into a bin that contains no item, we say that  $ALG$  *opens* the bin. The termination conditions of the three subroutines are as follows: SUBROUTINE1 finishes when it has given  $\text{Length}$  items to  $ON$ , SUBROUTINE2 finishes when  $ON$  has opened new  $\text{Length}$  bins, and SUBROUTINE3 finishes when  $ON$  has created  $\text{Length}$  bins with two items.

Before giving their pseudocodes, we define the function  $f$  used in these subroutines: for any  $x, y \in (0, 1]$  with  $x < y$ ,  $f(x, y) = (x + y)/2$ . (Indeed,  $f$  can be any function that maps  $x$  and  $y$  to a value between  $x$  and  $y$ .)

---

**SUBROUTINE1( $ON$ ,  $\text{Min}$ ,  $\text{Max}$ ,  $\text{Length}$ ):**

---

**Step 1.**  $a_1 := f(\text{Min}, \text{Max})$ ,  $\text{tmpMax} := \text{Max}$ ,  $\text{tmpMin} := \text{Min}$ , and  $i := 1$ .

**Step 2.** Give an item  $b_i$  of size  $a_i$ , and do the following according to  $ON$ 's action.

**Case 2.1.** If  $ON$  opens a bin and puts  $b_i$  into it,

$$a_{i+1} := f(\text{tmpMin}, a_i) \text{ and } \text{tmpMax} := a_i.$$

**Case 2.2.** Otherwise,

$$a_{i+1} := f(a_i, \text{tmpMax}) \text{ and } \text{tmpMin} := a_i.$$

**Step 3.** If  $\text{Length} = i$ , then return  $\text{tmpMin}$ . Otherwise,  $i := i + 1$ , and go to Step 2.

---

**SUBROUTINE2( $ON$ ,  $\text{Min}$ ,  $\text{Max}$ ,  $\text{Length}$ ):**

---

**Step 1.**  $a_1 := f(\text{Min}, \text{Max})$ ,  $\text{tmpMax} := \text{Max}$ ,  $\text{tmpMin} := \text{Min}$ , and  $i := 1$ .

**Step 2.** Give  $ON$  an item  $b_i$  of size  $a_i$ , and do the following according to  $ON$ 's action.

**Case 2.1.** If  $ON$  opens a bin and puts  $b_i$  into it, then

$$a_{i+1} := f(\text{tmpMin}, a_i) \text{ and } \text{tmpMax} := a_i.$$

**Case 2.2.** Otherwise,

$$a_{i+1} := f(a_i, \text{tmpMax}) \text{ and } \text{tmpMin} := a_i.$$

**Step 3.** If the number of bins that were opened by  $ON$  at Case 2.1 is  $\text{Length}$ , then return  $\text{tmpMin}$ . Otherwise,  $i := i + 1$ , and go to Step 2.

---

**SUBROUTINE3( $ON$ ,  $\text{Min}$ ,  $\text{Max}$ ,  $\text{Length}$ ):**

---

**Step 1.**  $a_1 := f(\text{Min}, \text{Max})$ ,  $\text{tmpMax} := \text{Max}$ ,  $\text{tmpMin} := \text{Min}$ , and  $i := 1$ .

**Step 2.** Give  $ON$  an item  $b_i$  of size  $a_i$ , and do the following according to  $ON$ 's action.

**Case 2.1.** If  $ON$  opens a bin and puts  $b_i$  into it, then

$$a_{i+1} := f(\text{tmpMin}, a_i) \text{ and } \text{tmpMax} := a_i.$$

**Case 2.2.** Otherwise,

$$a_{i+1} := f(a_i, \text{tmpMax}) \text{ and } \text{tmpMin} := a_i.$$

**Step 3.** If the number of bins with two items both of which are given at Step 2 is  $\text{Length}$ , then return  $\text{tmpMin}$ . Otherwise,  $i := i + 1$ , and go to Step 2.

---

The purpose of these subroutines is to construct a sequence that has the following property. The proof of the lemma will be provided in the full version.

**Lemma 1.** Suppose that SUBROUTINE1 (SUBROUTINE2, SUBROUTINE3, respectively) is called with some  $ON$ ,  $\text{Max}$ , and  $\text{Min}$  with  $\text{Max} > \text{Min}$ . Let  $\beta_j$  be the size of the  $j (= 1, \dots, n)$ -th item that is put into a bin at Case 2.1, and let  $\gamma_{j'}$  be the size of the  $j' (= 1, \dots, m)$ -th item that is put into a bin at Case 2.2. Denote  $\beta_0 := \text{Max}$ ,  $\gamma_0 := \text{Min}$ , and  $t_{\min}$  ( $t_{\max}$ ) the value of  $\text{tmpMin}$  ( $\text{tmpMax}$ , respectively) at the moment when the subroutine returns. Then,  $\beta_0 > \beta_1 > \dots > \beta_n = t_{\max} > t_{\min} = \gamma_m > \dots > \gamma_1 > \gamma_0$ .

Now we are ready to describe the main routines any of which the adversary chooses as its strategy. We remark here that ROUTINE1 outputs an equivalent

sequence to one used for getting a lower bound for  $k = 2$  in [1]. In that analysis the competitiveness of an online algorithm depends on how it packs the items that correspond to Step 1. Our analysis, in addition, examines how to deal with the items given in Step 3 and Step 4 of ROUTINE2 and 3.

The variables  $t$ ,  $b$ ,  $s$ ,  $x$ ,  $y$ ,  $u$ ,  $z$ ,  $w$ , and  $v$  appearing in the pseudocodes are used both for the execution of the routine and for the later analysis. “#” stands for “the number of”.

---

**ROUTINE1( $ON$ ,  $Length$ ):**


---

**Step 1.** Call SUBROUTINE1( $ON$ ,  $\frac{1}{10}$ ,  $\frac{1}{9}$ ,  $Length$ ), and  $t :=$  (the return value).

Then,  $\frac{x}{2} :=$  (# bins with two items), and  $y :=$  (# bins with exactly one item).

**Step 2.** Give  $ON \frac{x}{2}$  items of size  $1 - t$ .

---

**ROUTINE2( $ON$ ,  $Length$ ):**


---

**Step 1.** Call SUBROUTINE1( $ON$ ,  $\frac{1}{10}$ ,  $\frac{1}{9}$ ,  $Length$ ), and  $t :=$  (the return value).

Then,  $\frac{x}{2} :=$  (# bins with two items), and  $y :=$  (# bins with exactly one item).

**Step 2.** Give  $ON \frac{x}{2}$  items of size  $1 - t$ .

**Step 3.** Call SUBROUTINE2( $ON$ ,  $\frac{4}{5}$ ,  $\frac{7}{8}$ ,  $y + \frac{x}{2}$ ), and  $b :=$  (the return value).

Then,  $u :=$  (# bins with one item given in Step 1 and one given in Step 3).

**Step 4.** Call SUBROUTINE1( $ON$ ,  $\frac{1}{6}$ ,  $1 - b$ ,  $u$ ).

Then,  $z :=$  (# bins with one item given in Step 1 and one given in Step 4).

---

**ROUTINE3( $ON$ ,  $Length$ ):**


---

**Step 1.** Call SUBROUTINE1( $ON$ ,  $\frac{1}{10}$ ,  $\frac{1}{9}$ ,  $Length$ ), and  $t :=$  (the return value).

Then,  $\frac{x}{2} :=$  (# bins with two items), and  $y :=$  (# bins with exactly one item).

**Step 2.** Give  $ON \frac{x}{2}$  items of size  $1 - t$ .

**Step 3.** Call SUBROUTINE2( $ON$ ,  $\frac{4}{5}$ ,  $\frac{7}{8}$ ,  $y + \frac{x}{2}$ ), and  $b :=$  (the return value).

Then,  $u :=$  (# bins with one item given in Step 1 and one given in Step 3).

**Step 4.** Call SUBROUTINE3( $ON$ ,  $\frac{1}{6}$ ,  $1 - b$ ,  $u$ ), and  $s :=$  (the return value).

Then,  $z + w :=$  (# bins with one item given in Step 1 and one given in Step 4), and  $v :=$  (# bins with exactly one item given in Step 4).

**Step 5.** Give  $ON u + z + w$  items of size  $1 - s$ .

---

For an arbitrary online algorithm  $ALG$  and a positive integer  $Length$ , let ROUTINE1, 2, and 3 run and generate sequences of items  $\sigma_1$ ,  $\sigma_2$ , and  $\sigma_3$ , respectively. What should be remarked upon here is that  $\sigma_1$  is a prefix of  $\sigma_2$  and  $\sigma_2$  is a prefix of  $\sigma_3$ . (This verifies the consistency of the variables  $z(\geq 0)$  and  $w(\geq 0)$  set in ROUTINE2 and 3.) Now we see what items are included in the longest sequence  $\sigma_3$ . According to the values  $t$ ,  $b$ , and  $s$  determined through the execution of ROUTINE3, we classify all items into the following eight categories:

- $t_-$ -items, those which are of size in  $(\frac{1}{10}, t]$  and given in Step 1,
- $t_+$ -items, those which are of size in  $(t, \frac{1}{9})$  and given in Step 1,
- $(1 - t)$ -items, those which are of size  $(1 - t)$  and given in Step 2,
- $b_-$ -items, those which are of size in  $(\frac{4}{5}, b]$  and given in Step 3,

- $\mathbf{b}_+$ -items, those which are of size in  $(b, \frac{7}{8})$  and given in Step 3,
- $\mathbf{s}_-$ -items, those which are of size in  $(\frac{1}{6}, s]$  and given in Step 4,
- $\mathbf{s}_+$ -items, those which are of size in  $(s, 1 - b)$  and given in Step 4, and
- $(1 - \mathbf{s})$ -items, those which are of size  $(1 - s)$  and given in Step 5.

Lemma 1 clarifies the magnitude relation among items given in each subroutine. Together with the categorization above, we have the next fact:

**Fact.** *In the execution of SUBROUTINE1 (2, 3, respectively) called by ROUTINE3, an item that is put into a bin at Case 2.1 is classified as a  $\mathbf{t}_+$ -item ( $\mathbf{b}_+$ -item,  $\mathbf{s}_+$ -item, respectively), while one that is put into a bin at Case 2.2 is classified as a  $\mathbf{t}_-$ -item ( $\mathbf{b}_-$ -item,  $\mathbf{s}_-$ -item, respectively).*

The lemma below counts the numbers of non-empty bins of  $ALG$  and  $OPT$ . See Figure 1 for the packings. The proof will appear in the full version.

**Lemma 2.** *For  $ALG$ ,  $OPT$ , and  $(x, y, u, z, w, v)$  determined by each of ROUTINE1, 2, and 3, it holds that:*

$$\begin{aligned} \text{Length} &= x + y, \\ C_{ALG}(\sigma_1) &= x + y, \\ C_{OPT}(\sigma_1) &= \frac{1}{2}x + \left\lceil \frac{1}{4}x + \frac{1}{2}y \right\rceil \leq \frac{3}{4}x + \frac{1}{2}y + \frac{1}{2}, \\ C_{ALG}(\sigma_2) &\geq \frac{3}{2}x + 2y + \frac{1}{2}(u - z), \\ C_{OPT}(\sigma_2) &= x + y + u, \\ C_{ALG}(\sigma_3) &\geq \frac{3}{2}x + y + 3u + v + 2z + 2w, \\ C_{OPT}(\sigma_3) &= x + y + 2u + z + w + \left\lceil \frac{1}{2}v \right\rceil \leq x + y + 2u + z + w + \frac{1}{2}v + \frac{1}{2}. \end{aligned}$$

The next lemma is the heart of our analysis, which follows from Lemmas 4 and 5. The proof of Lemma 4 is omitted here.

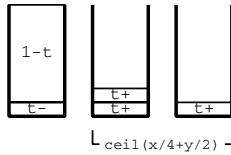
**Lemma 3.** *For an arbitrary online algorithm  $ALG$  and any  $\varepsilon > 0$ , there exists a positive integer  $\text{Length}$  such that: Let ROUTINE1, 2, and 3 run with  $ALG$  and  $\text{Length}$  as parameters, and generate sequences of items  $\sigma_1$ ,  $\sigma_2$ , and  $\sigma_3$ , respectively. Then, it follows that*

$$\max \left\{ \frac{C_{ALG}(\sigma_1)}{C_{OPT}(\sigma_1)}, \frac{C_{ALG}(\sigma_2)}{C_{OPT}(\sigma_2)}, \frac{C_{ALG}(\sigma_3)}{C_{OPT}(\sigma_3)} \right\} \geq \bar{r} - \varepsilon, \quad (1)$$

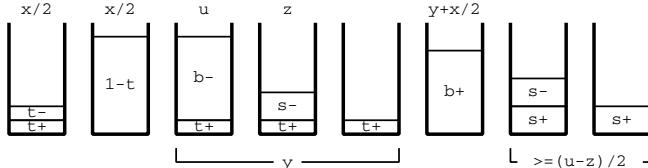
where  $\bar{r} (\approx 1.42764)$  is the root of the cubic equation  $2r^3 - 17r^2 + 30r - 14 = 0$  which lies between  $\frac{4}{3}$  and  $\frac{3}{2}$ .

C\_ALG(sigma1) =  $x + y$   
 $x/2 \quad x/2 \quad y$

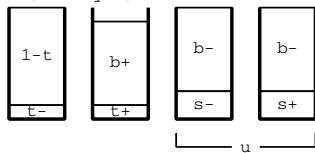
C\_OPT(sigma1) =  $x/2 + \text{ceil}(x/4 + y/2) \leq (3/4)x + y/2 + 1/2$   
 $x/2$



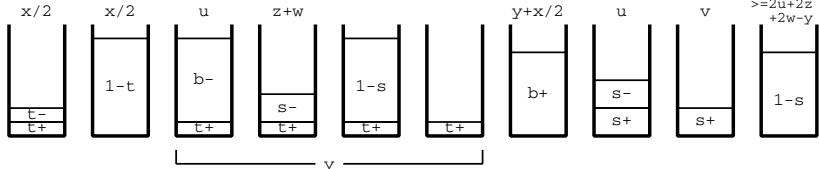
C\_ALG(sigma2)  $\geq (3/2)x + 2y + (u-z)/2$   
 $x/2 \quad x/2 \quad u \quad z$



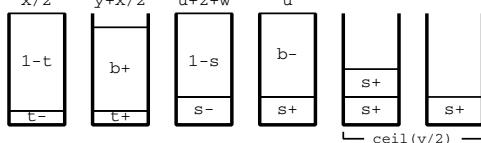
C\_OPT(sigma2) =  $x + y + u$   
 $x/2 \quad y+x/2$



C\_ALG(sigma3)  $\geq (3/2)x + y + 3u + v + 2z + 2w$   
 $x/2 \quad x/2 \quad u \quad z+w$



C\_OPT(sigma3) =  $x + y + 2u + z + w + \text{ceil}(v/2) \leq x + y + 2u + z + w + v/2 + 1/2$   
 $x/2 \quad y+x/2 \quad u+z+w \quad u$



**Fig. 1.** Packings by an arbitrary online algorithm and an optimal offline algorithm for the input sequences generated by ROUTINE1, 2, and 3. The numbers above (or below) bins indicate the number of bins belonging to that type.

**Lemma 4.** For any  $\varepsilon > 0$ , there exists a positive integer Length such that for any nonnegative integers  $x, y, u, z, w$ , and  $v$  with  $x + y = \text{Length}$ ,

$$\frac{x+y}{\frac{3}{4}x+\frac{1}{2}y} - \frac{x+y}{\frac{3}{4}x+\frac{1}{2}y+\frac{1}{2}} \leq \varepsilon, \quad (2)$$

$$\frac{\frac{3}{2}x+y+3u+2z+2w+v}{x+y+2u+z+w+\frac{1}{2}v} - \frac{\frac{3}{2}x+y+3u+2z+2w+v}{x+y+2u+z+w+\frac{1}{2}v+\frac{1}{2}} \leq \varepsilon. \quad (3)$$

**Lemma 5.** For any nonnegative integers  $x, y, u, z, w$ , and  $v$  with  $x + y > 0$ ,

$$\max \left\{ \frac{x+y}{\frac{3}{4}x+\frac{1}{2}y}, \frac{\frac{3}{2}x+2y+\frac{1}{2}(u-z)}{x+y+u}, \frac{\frac{3}{2}x+y+3u+v+2z+2w}{x+y+2u+z+w+\frac{1}{2}v} \right\} \geq \bar{r}. \quad (4)$$

*Proof.* The proof is done by contradiction. Assume the lemma to be false. Then, all of the operands of the max operation in (4) can fall below  $\bar{r}$  in the same time. That is to say, there exists a tuple of nonnegative integers  $(x, y, u, z, w, v)$  with  $x + y > 0$  such that

$$f_1 := x + y - \bar{r} \left( \frac{3}{4}x + \frac{1}{2}y \right) < 0, \quad (5)$$

$$f_2 := \frac{3}{2}x + 2y + \frac{1}{2}(u-z) - \bar{r}(x+y+u) < 0, \quad (6)$$

$$f_3 := \frac{3}{2}x + y + 3u + v + 2z + 2w - \bar{r} \left( x + y + 2u + z + w + \frac{1}{2}v \right) < 0. \quad (7)$$

In what follows we show that there is no such  $(x, y, u, z, w, v)$ . Specifically, we derive an inequality that does not contain either  $x$ ,  $y$ , or  $u$  from the inequalities (5), (6), and (7). We then claim that there do not exist  $z$ ,  $w$ , and  $v$  which satisfy the derived inequality.

Recall  $\frac{4}{3} < \bar{r} < \frac{3}{2}$ . Noting that  $3 - 2\bar{r}$  and  $2\bar{r} - 1$  are both positive, we have an inequality without  $u$  from (6) and (7).

$$\begin{aligned} f_4 &:= 4(3 - 2\bar{r})f_2 + 2(2\bar{r} - 1)f_3 \\ &= (-2\bar{r}^2 + 5\bar{r} - 2)v + (-4\bar{r}^2 + 10\bar{r} - 4)w + (4\bar{r}^2 - 16\bar{r} + 15)x \\ &\quad + (4\bar{r}^2 - 22\bar{r} + 22)y + (-4\bar{r}^2 + 14\bar{r} - 10)z \\ &< 0. \end{aligned}$$

(Please see that the elimination is done so that the resulting inequality sign makes sense.) Next, let us eliminate  $x$ . The coefficient of  $x$  in the above inequality  $4\bar{r}^2 - 16\bar{r} + 15 = (2\bar{r} - 5)(2\bar{r} - 3)$  is confirmed to be positive. Together with positivity of  $3\bar{r} - 4$ , we eliminate  $x$  using (5).

$$\begin{aligned}
f_5 &:= 4(4\bar{r}^2 - 16\bar{r} + 15)f_1 + (3\bar{r} - 4)f_4 \\
&= (-6\bar{r}^3 + 23\bar{r}^2 - 26\bar{r} + 8)(v + 2w) + \\
&\quad 2(2\bar{r}^3 - 17\bar{r}^2 + 30\bar{r} - 14)y + 2(-6\bar{r}^3 + 29\bar{r}^2 - 43\bar{r} + 20)z \\
&= (-6\bar{r}^3 + 23\bar{r}^2 - 26\bar{r} + 8)(v + 2w) + 2(-6\bar{r}^3 + 29\bar{r}^2 - 43\bar{r} + 20)z \\
&< 0.
\end{aligned}$$

The reason why  $y$  has gone is because  $\bar{r}$  is a root of  $2r^3 - 17r^2 + 30r - 14 = 0$ .  $\frac{4}{3} < \bar{r} < \frac{3}{2}$  leads to that both  $(-6\bar{r}^3 + 23\bar{r}^2 - 26\bar{r} + 8)$  and  $(-6\bar{r}^3 + 29\bar{r}^2 - 43\bar{r} + 20)$  are positive. Therefore, for fulfilling  $f_5 < 0$ , either  $z$ ,  $w$ , or  $v$  should be negative. This contradicts the assumption that  $z$ ,  $w$ , and  $v$  are all nonnegative.  $\square$

Our new lower bound is obtained almost as a corollary from Lemma 3.

**Theorem 1.** *Any online algorithm for the online 2-item bin packing problem has an asymptotic competitive ratio of at least  $\bar{r}$ , where  $\bar{r} (\approx 1.42764)$  is the root of the cubic equation  $2r^3 - 17r^2 + 30r - 14 = 0$  which lies between  $\frac{4}{3}$  and  $\frac{3}{2}$ .*

### 3 Lower Bounds for $k \geq 4$

We propose an approach for deriving a lower bound of the online  $k$ -item bin packing problem for each  $k \geq 4$ , expanding the method of van Vliet [13] for the problem without a cardinality constraint. His method was to solve a linear program in which variables represent the packings by an arbitrary online algorithm given some patterns of input sequences. We illustrate how to embed a cardinality constraint into the linear program.

Some existing lower bounds for the problem without a cardinality constraint, such as [14,13,2], can be interpreted as lower bounds for the online  $k$ -item bin packing for some ranges of  $k$ ; if the possible item size is restricted to be at least  $s$ , then the problem can be seen as the online  $k$ -item bin packing for  $k \geq \lfloor \frac{1}{s} \rfloor$ , since there is no chance that more than  $\lfloor \frac{1}{s} \rfloor$  items are packed together. Such results include: A lower bound of  $\frac{4}{3}$  for  $4 \leq k \leq 5$  [13],  $\frac{3}{2}$  for  $6 \leq k \leq 41$  [14], and  $\frac{217}{141}$  for  $42 \leq k$  [13]. See Table 1 in Section 1. Note that although the paper [13] does not provide the value of  $\frac{4}{3}$  explicitly, it is given just by slightly changing the settings of his method. In the derivation of these results, it is not assumed that an algorithm packs items so that the cardinality constraint is kept. After the reformulation of a linear program, we set  $k < \lfloor \frac{1}{s} \rfloor$  and try to obtain a better lower bound.

We first give our new formulation with the cardinality constraint. We are given a tuple of item sizes  $(s_1, \dots, s_l)$  with  $\sum_{i=1}^l s_i \leq 1$ . Set  $N$  a positive integer divisible by  $k$  and  $\lfloor \frac{1}{\sum_{h=i}^l s_h} \rfloor$  for all  $1 \leq i \leq l$ . Let  $L_i$  be a sequence of  $N$  items of size  $s_i$  for each  $1 \leq i \leq l$ . The adversary issues any of  $L_l \cdots L_i$  ( $1 \leq i \leq l$ ).

We denote by a vector  $(t_1, \dots, t_l)^T$  a packing of a bin that consists of  $t_i$  items of size  $s_i$  for  $1 \leq i \leq l$ . Any packing has to satisfy the following constraints: (i) the capacity constraint  $\sum_{i=1}^l t_i s_i \leq 1$ , (ii) the cardinality constraint  $\sum_{i=1}^l t_i \leq$

$k$ , and (iii) the constraint that only non-empty bins are taken into account  $\sum_{i=1}^l t_i > 0$ .

Sort all feasible packings in a lexicographical order with an entry of an item size with a larger index having a bigger priority. For example, for  $(s_1, s_2, s_3) = (\frac{1}{2} + \varepsilon, \frac{1}{3} + \varepsilon, \frac{1}{7} + \varepsilon)$  and  $k = 4$ , we have  $(1, 0, 0)^T, (0, 1, 0)^T, (1, 1, 0)^T, \dots, (1, 0, 3)^T, (0, 1, 3)^T, (0, 0, 4)^T$ . Denote by  $t_{i,j}$  the  $i$ -th entry of the  $j$ -th packing in the sorted list. The set of  $t_{i,j}$ 's is regarded as a matrix. For the above example,

$$(t_{i,j}) = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 2 & 0 & 0 & 1 & 1 & 2 & 0 & 0 & 1 & 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 4 \end{pmatrix}.$$

Let  $m$  be the number of feasible packings, which is 17 for the example.

Fix an online algorithm  $ALG$  arbitrarily. Suppose that given the input sequence  $L_l \cdots L_1$ ,  $ALG$  creates  $n_j$  bins with the  $j$ -th packing (i.e.,  $(t_{1,j}, \dots, t_{l,j})^T$ ). Define  $p_i$  as the index of the first column that has a non-zero entry in the  $i$ -th row of the matrix  $(t_{i,j})$ . Then it holds that for  $i \geq 2$  a packing before the  $p_i$ -th is one that skips all of items of size  $s_l, \dots, s_i$  and begins packing from  $s_{i-1}$ , and that  $p_1 = 1$ . For the above example,  $p_1 = 1$ ,  $p_2 = 2$ , and  $p_3 = 5$ . Thus, we can describe the total number of non-empty bins of  $ALG$  for  $L_l \cdots L_i$  ( $1 \leq i \leq l$ ) as

$$C_{ALG}(L_l \cdots L_i) = \sum_{j=p_i}^m n_j. \quad (8)$$

The total number of non-empty bins of an optimal offline algorithm  $OPT$  is bounded by a simple but nontrivial lemma. We will provide the proof in the full version.

**Lemma 6.** *For  $1 \leq i \leq l$ ,  $C_{OPT}(L_l \cdots L_i) \leq \max\{\frac{N}{\lfloor \frac{1}{\sum_{h=i}^l s_h} \rfloor}, \frac{N(l-i+1)}{k}\}$ .*

As a matter of course, the whole set of bins created by  $ALG$  for  $L_l \cdots L_1$  contains  $N$  items of size  $s_i$  for each  $1 \leq i \leq l$ . This fact is described as  $\sum_{j=1}^m t_{i,j} n_j = N$  for all  $1 \leq i \leq l$ . Note that as long as this equation holds, the packings for  $L_l \cdots L_i$  ( $1 \leq i \leq l-1$ ) are consistent as well. For later formulation, we rewrite this as

$$\sum_{j=1}^m \frac{t_{i,j} n_j}{N} - 1 = 0, \quad 1 \leq i \leq l. \quad (9)$$

The asymptotic competitive ratio  $R_{ALG}$  is asymptotically lower-bounded by  $R$  such that

$$\frac{C_{ALG}(L_l \cdots L_i)}{C_{OPT}(L_l \cdots L_i)} - R \leq 0, \quad 1 \leq i \leq l. \quad (10)$$

A sufficient condition for (10) with slack variables  $(u_1, \dots, u_l)$  is

$$\min\left\{\left\lfloor \frac{1}{\sum_{h=i}^l s_h} \right\rfloor, \frac{k}{l-i+1}\right\} \sum_{j=p_i}^m \frac{n_j}{N} + u_i - R = 0, \quad 1 \leq i \leq l \quad (11)$$

**Table 2.** Our new lower bounds for each  $4 \leq k \leq 45$ . Bold font indicates improvement.

| $k$ | lower bound                         | $k$ | lower bound                         | $k$ | lower bound                           |
|-----|-------------------------------------|-----|-------------------------------------|-----|---------------------------------------|
| 4   | $\frac{3}{2} (= 1.5)$               | 18  | $\frac{93}{61} (\approx 1.52459)$   | 32  | $\frac{496}{323} (\approx 1.53560)$   |
| 5   | $\frac{25}{17} (\approx 1.47058)$   | 19  | $\frac{171}{112} (\approx 1.52678)$ | 33  | $\frac{341}{222} (\approx 1.53603)$   |
| 6   | $\frac{3}{2} (= 1.5)$               | 20  | $\frac{315}{206} (\approx 1.52912)$ | 34  | $\frac{527}{343} (\approx 1.53644)$   |
| 7   | $\frac{3}{2} (= 1.5)$               | 21  | $\frac{26}{17} (\approx 1.52941)$   | 35  | $\frac{1085}{706} (\approx 1.53682)$  |
| 8   | $\frac{3}{2} (= 1.5)$               | 22  | $\frac{341}{223} (\approx 1.52914)$ | 36  | $\frac{186}{121} (\approx 1.53719)$   |
| 9   | $\frac{3}{2} (= 1.5)$               | 23  | $\frac{713}{466} (\approx 1.53004)$ | 37  | $\frac{1147}{746} (\approx 1.53753)$  |
| 10  | $\frac{80}{53} (\approx 1.50943)$   | 24  | $\frac{124}{81} (\approx 1.53086)$  | 38  | $\frac{589}{383} (\approx 1.53785)$   |
| 11  | $\frac{44}{29} (\approx 1.51724)$   | 25  | $\frac{775}{506} (\approx 1.53162)$ | 39  | $\frac{403}{262} (\approx 1.53816)$   |
| 12  | $\frac{66}{43} (\approx 1.53488)$   | 26  | $\frac{403}{263} (\approx 1.53231)$ | 40  | $\frac{20}{13} (\approx 1.53846)$     |
| 13  | $\frac{26}{17} (\approx 1.52941)$   | 27  | $\frac{279}{182} (\approx 1.53296)$ | 41  | $\frac{1271}{826} (\approx 1.53874)$  |
| 14  | $\frac{441}{289} (\approx 1.52595)$ | 28  | $\frac{434}{283} (\approx 1.53356)$ | 42  | $\frac{1519}{993} (\approx 1.52970)$  |
| 15  | $\frac{315}{206} (\approx 1.52912)$ | 29  | $\frac{899}{586} (\approx 1.53412)$ | 43  | $\frac{9331}{6098} (\approx 1.53017)$ |
| 16  | $\frac{624}{409} (\approx 1.52567)$ | 30  | $\frac{155}{101} (\approx 1.53465)$ | 44  | $\frac{4774}{3119} (\approx 1.53061)$ |
| 17  | $\frac{527}{346} (\approx 1.52312)$ | 31  | $\frac{961}{626} (\approx 1.53514)$ | 45  | $\frac{3255}{2126} (\approx 1.53104)$ |

for some  $u_i \geq 0$  ( $1 \leq i \leq l$ ). The derivation follows from (8) and  $\frac{N}{C_{OPT}(L_l \cdots L_i)} \geq \min\{\lfloor \frac{1}{\sum_{h=i}^l s_h} \rfloor, \frac{k}{l-i+1}\}$  obtained from Lemma 6.

The problem of finding the minimum  $R$  that satisfies (9) and (11) is formulated as a mathematical program ( $\mathcal{P}_N$ ) with a  $2l \times (m+l+1)$ -matrix  $A = (a_{i,j})$  and vectors  $\mathbf{x}$ ,  $\mathbf{b}$ , and  $\mathbf{c}$  as below.

$$a_{i,j} = \begin{cases} t_{i,j}, & 1 \leq i \leq l, 1 \leq j \leq m; \\ 0, & 1 \leq i \leq l, m+1 \leq j \leq m+l+1; \\ 0, & l+1 \leq i \leq 2l, 1 \leq j \leq p_{i-l}-1; \\ \min\left\{\lfloor \frac{1}{\sum_{h=i}^l s_h} \rfloor, \frac{k}{l-i+1}\right\}, & l+1 \leq i \leq 2l, p_{i-l} \leq j \leq m; \\ \delta_{i-l,j-m}, & l+1 \leq i \leq 2l, m+1 \leq j \leq m+l; \\ -1, & l+1 \leq i \leq 2l, j = m+l+1. \end{cases}$$

$$\mathbf{x} = \left( \frac{n_1}{N}, \dots, \frac{n_m}{N}, u_1, \dots, u_l, R \right)^T, \mathbf{b} = \left( \overbrace{1, \dots, 1}^l, \overbrace{0, \dots, 0}^l \right)^T, \mathbf{c} = \left( \overbrace{0, \dots, 0}^{m+l}, 1 \right)^T$$

$$(\mathcal{P}_N) \quad \text{minimize } \mathbf{c}^T \mathbf{x}$$

$$\text{subject to } A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0, \mathbf{x} = \left( \frac{n_1}{N}, \dots, \frac{n_m}{N}, u_1, \dots, u_l, R \right)$$

$$(n_1, \dots, n_m) \in \mathbb{Z}^m, (u_1, \dots, u_l, R) \in \mathbb{R}^{l+1}$$

Here  $\delta_{i,j}$  is Kronecker delta (if  $i = j$ ,  $\delta_{i,j} = 1$ ; otherwise,  $\delta_{i,j} = 0$ ).

In  $A\mathbf{x} = \mathbf{b}$ , the first  $l$  rows correspond to (9), while the  $(l+1)$ -th to  $2l$ -th rows correspond to (11).  $\delta_{i-l,j-m}$  lets the slack variable  $u_i$  appear in the equation of the  $(l+i)$ -th row. The objective function is  $\mathbf{c}^T \mathbf{x} = R$ . Note that  $A$ ,  $\mathbf{b}$ , and  $\mathbf{c}$  are independent of the choice of  $N$ .

Apparently, the optimal value of the following linear program  $(\mathcal{P})$  is a lower bound on the optimal value of  $(\mathcal{P}_N)$ .

$$\begin{aligned} (\mathcal{P}) \quad & \text{minimize } \mathbf{c}^T \mathbf{x} \\ & \text{subject to } A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0, \mathbf{x} \in \mathbb{R}^{m+l+1} \end{aligned}$$

The next theorem provides a lower bound for each  $4 \leq k \leq 45$ . The reason why we do not mention  $k \geq 46$  is simply because of space limitation. Note that as long as the computer power is available, one can calculate a lower bound for arbitrary  $k$  using our method. The proof is left to the full version.

**Theorem 2.** *For each  $4 \leq k \leq 45$ , any online algorithm for the online  $k$ -item bin packing problem has an asymptotic competitive ratio of at least the value in Table 2.*

One can see that the new lower bounds for some  $k$ , such as  $k = 5$  or  $13$ , are lower than that for smaller  $k$ . We believe, however, that the matching upper and lower bound increases with respect to  $k$  and approaches that for the problem without a cardinality constraint. The anomaly suggests a limit of our method for some values of  $k$ . It is an interesting open problem to construct a better scheme for a lower bound for arbitrary  $k$ .

## References

1. Babel, L., Chen, B., Kellerer, H., Kotov, V.: Algorithms for on-line bin-packing problems with cardinality constraints. *Discrete Applied Mathematics* 143(1-3), 238–251 (2004)
2. Balogh, J., Békési, J., Galambos, G.: New lower bounds for certain classes of bin packing algorithms. *Theor. Comput. Sci.* 440-441, 1–13 (2012)
3. Borodin, A., El-Yaniv, R.: *Online Computation and Competitive Analysis*. Cambridge University Press (1998)
4. Caprara, A., Kellerer, H., Pferschy, U.: Approximation schemes for ordered vector packing problems. *Naval Research Logistics* 50(1), 58–69 (2003)
5. Epstein, L.: Online bin packing with cardinality constraints. *SIAM J. Discrete Math.* 20(4), 1015–1030 (2006)
6. Epstein, L., Levin, A.: AFPTAS results for common variants of bin packing: A new method for handling the small items. *SIAM Journal on Optimization* 20(6), 3121–3145 (2010)
7. Kellerer, H., Pferschy, U.: Cardinality constrained bin-packing problems. *Annals of Operations Research* 92, 335–348 (1999)
8. Krause, K.L., Shen, V.Y., Schwetman, H.D.: Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems. *J. ACM* 22(4), 522–550 (1975)

9. Krause, K.L., Shen, V.Y., Schwetman, H.D.: Errata: “Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems”. *J. ACM* 24(3), 527 (1977)
10. Ramanan, P.V., Brown, D.J., Lee, C.C., Lee, D.T.: On-line bin packing in linear time. *J. Algorithms* 10(3), 305–326 (1989)
11. Seiden, S.S.: On the online bin packing problem. *J. ACM* 49(5), 640–671 (2002)
12. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. *Commun. ACM* 28(2), 202–208 (1985)
13. van Vliet, A.: An improved lower bound for on-line bin packing algorithms. *Inf. Process. Lett.* 43(5), 277–284 (1992)
14. Yao, A.C.: New algorithms for bin packing. *J. ACM* 27(2), 207–227 (1980)

# Parameterized Complexity of Flood-Filling Games on Trees

Uéverton dos Santos Souza<sup>1</sup>, Fábio Protti<sup>1</sup>, and Maise Dantas da Silva<sup>2</sup>

<sup>1</sup> Institute of Computing

<sup>2</sup> PURO/ICT

Fluminense Federal University, Niterói, RJ, Brazil

{usuouza,fabio}@ic.uff.br,

maisedantas@id.uff.br

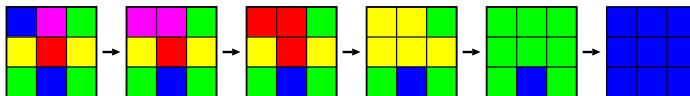
**Abstract.** This work presents new results on flood-filling games, Flood-It and Free-Flood-It, in which the player aims to make the board monochromatic with a minimum number of flooding moves. As for many colored graph problems, Flood-filling games have relevant interpretations in bioinformatics. The standard versions of Flood-It and Free-Flood-It are played on  $n \times m$  grids. In this paper we analyze the complexity of these games when played on trees. We prove that Flood-It remains NP-hard on trees whose leaves are at distance at most  $d = 2$  from the pivot, and that Flood-It is in FPT when parameterized by the number of colors  $c$  in such trees (for any constant  $d$ ). We also show that Flood-It on trees and Restricted Shortest Common Supersequence (RSCS) are analogous problems, in the sense that they can be translated from one to another, keeping complexity features; this implies that Flood-It on trees inherits several complexity results already proved for RSCS, such as some interesting FPT and W[1]-hard cases. We introduce a new variant of Flood-It, called Multi-Flood-It, where each move of the game is played on various pivots. We also present a general framework for reducibility from Flood-It to Free-Flood-It, by defining a special graph operator  $\psi$  such that Flood-It played on a graph class  $\mathcal{F}$  is reducible to Free-Flood-It played on the image of  $\mathcal{F}$  under  $\psi$ . An interesting particular case occurs when  $\mathcal{F}$  is closed under  $\psi$ . Some NP-hard cases for Free-Flood-It on trees can be derived using this approach. We conclude by showing some results on parameterized complexity for Free-Flood-It played on pc-trees (phylogenetic colored trees). We prove that some results valid for Flood-It on pc-trees can be inherited by Free-Flood-It on pc-trees, using another type of reducibility framework.

**Keywords:** Combinatorial Games, Fixed Parameter Tractability, Graph Algorithms, NP-hardness, W[1]-hardness.

## 1 Introduction

Flood-It is a one-player combinatorial game, originally played on a colored board consisting of an  $n \times m$  grid, where each tile of the board has an initial color from a

fixed color set. In the classic game, two tiles are *neighboring* tiles if they lie in the same row (resp. column) and in consecutive columns (resp. rows). A sequence  $C$  of tiles is a *path* when every pair of consecutive tiles in  $C$  is formed by neighboring tiles. A *monochromatic path* is a path in which all the tiles have the same color. Two tiles  $a$  and  $b$  are  $m$ -*connected* when there is a monochromatic path between them. In Flood-It, a move consists of assigning a new color  $c$  to the top left tile  $p$  (the *pivot*) and also to all the tiles  $m$ -connected to  $p$  immediately before the move. The objective of the game is to make the board monochromatic (“flood the board”) with the minimum number of moves. Figure 1 shows a sequence of moves to flood a  $3 \times 3$  grid.



**Fig. 1.** An optimal sequence of moves to flood a  $3 \times 3$  grid

A variation of Flood-It is Free-Flood-It, where the player can freely choose which tile will be the pivot of each move. In addition, these games can easily be generalized to be played on any graph with an initial coloring  $\omega$ .

Many complexity issues on Flood-It and Free-Flood-It have recently been investigated. In [1], Arthur, Clifford, Jalsenius, Montanaro, and Sach show that Flood-It and Free-Flood-It are NP-hard on  $n \times n$  grids colored with at least three colors. Meeks and Scott [19] prove that Free-Flood-It is solvable in polynomial time on  $1 \times n$  grids and on 2-colored graphs, and also that Flood-It and Free-Flood-It remain NP-hard on  $3 \times n$  grids colored with at least four colors. Up to the authors’ knowledge, the complexity of (Free-)Flood-It on  $3 \times n$  grids colored with three colors remains as an open question. Clifford, Jalsenius, Montanaro, and Sach present in [6] a polynomial-time algorithm for Flood-It on  $2 \times n$  grids. In [20], Meeks and Scott show that Free-Flood-It remains NP-hard on  $2 \times n$  grids. Lagoutte, Noual, and Thierry [16] shows that Flood-It is polynomially solvable on cycles, and Flood-It and Free-Flood-It are NP-hard on trees.

In this work we analyze the complexity of Flood-It and Free-Flood-It when played on trees. In Section 2, we prove that Flood-It remains NP-hard on trees whose leaves are at distance at most  $d = 2$  from the pivot, and that Flood-It is in FPT when parameterized by the number of colors  $c$  in such trees (for any constant  $d$ ). Also in Section 2 we show that Flood-It on trees and Restricted Shortest Common Supersequence (RSCS) are analogous problems, in the sense that they can be translated from one to another, keeping complexity features; this implies that Flood-It on trees inherits several complexity results already proved for RSCS, such as some interesting FPT and W[1]-hard cases. We conclude Section 2 by introducing a new variant of Flood-It, called Multi-Flood-It, where each move is played on a set of fixed pivots; we consider Multi-Flood-It played on trees where the pivots are the leaves, and derive some results on

its complexity. In Section 3, we present a general framework for reducibility from Flood-It to Free-Flood-It, by defining a special graph operator  $\psi$  such that Flood-It played on a graph class  $\mathcal{F}$  is reducible to Free-Flood-It played on the image of  $\mathcal{F}$  via  $\psi$ . An interesting particular case occurs when  $\mathcal{F}$  is closed under  $\psi$  (for instance, trees are closed under  $\psi$ ). Some NP-hard cases for Free-Flood-It on trees can be derived using this approach. We conclude Section 3 by showing some results on parameterized complexity for Free-Flood-It played on pc-trees (*phylogenetic colored trees*). A colored rooted tree is a pc-tree if no color occurs more than once in any path from the root to a leaf. We prove that some results valid for Flood-It on pc-trees can be inherited by Free-Flood-It on pc-trees, using another type of reducibility framework.

**Flood-Filling Games in Bioinformatics.** Since the 90's, an increasing number of papers on biological applications have been dealt with as combinatorial problems. Vertex-colored graph problems have several applications in bioinformatics [8]. The Colored Interval Sandwich Problem has applications in DNA physical mapping [10,12] and in perfect phylogeny [18]; vertex-recoloring problems appear in protein-protein interaction networks and phylogenetic analysis [5,21]; the Graph Motif Problem [8] was introduced in the context of metabolic network analysis [15]; intervalizing colored graphs [4] model DNA physical mapping [10]; and the Triangulating Colored Graph Problem [4] is polynomially equivalent to the Perfect Phylogeny Problem [13].

Flood-Filling games on colored graphs are related to many problems in bioinformatics. As shown in this paper, Flood-it played on trees is analogous to a restricted case of the Shortest Common Supersequence Problem [14]. Consequently, these games inherit from the Shortest Common Supersequence Problem many implications in bioinformatics, such as: microarray production [23], DNA sequence assembly [3], and a close relationship to multiple sequence alignment [24]. In addition, some disease spreading models, described in [2], work in a similar way to flood-filling games.

**Additional Definitions and Notation.** Neighboring tiles naturally correspond to neighboring vertices of a graph  $G$  representing the board; therefore, from now on, we use the term *vertex* instead of *tile*. A subgraph  $H$  of  $G$  is *adjacent* to a vertex  $v \in V(G)$  if  $v$  has a neighbor in  $V(H)$ . A *move* is a pair  $m = (p, c)$  where  $p$  is the *pivot* of  $m$  (the vertex chosen to have its color changed by  $m$ ), and  $c$  is the new color assigned to  $p$ ; in this case, we also say that color  $c$  is *played in move*  $m$ . In Flood-It all moves have the same pivot. A subgraph  $H$  is said to be *flooded* when  $H$  becomes monochromatic. A (free-)flooding is a sequence of moves in (Free-)Flood-It which floods  $G$  (the entire board). An optimal (free-)flooding is a flooding with minimum number of moves. A vertex  $v$  is *flooded by a move*  $m$  if the color of  $v$  is played in  $m$  and  $v$  becomes m-connected to new vertices after playing  $m$ . In Free-Flood-It, a move  $m = (p, c)$  is *played on subgraph*  $H$  if  $p \in V(H)$ . We denote by  $\Pi \propto^f \Pi'$  a reduction from a problem  $\Pi$  to a problem  $\Pi'$  via a computable function  $f$ .

## 2 Flood-It on Trees

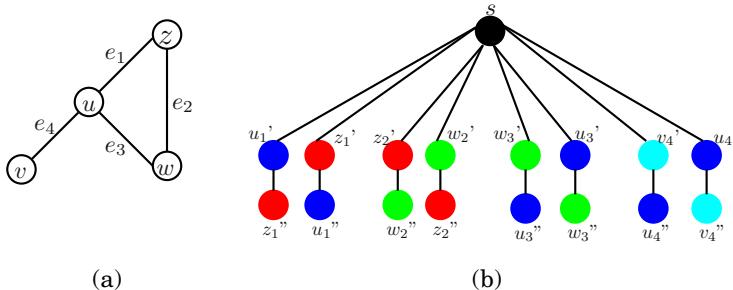
We start this section by remarking that Flood-It played on a tree is equivalent to Flood-It played on a rooted tree whose root is the pivot.

**Theorem 1.** *Flood-It remains NP-hard on trees whose leaves are at distance at most 2 from the pivot.*

**Proof.** The proof uses a reduction from the Vertex Cover problem. We show that there is a vertex cover of size  $k$  in a graph  $G$  if and only if there is a flooding with  $n+k$  moves in the associated tree  $T$ . Given a graph  $G = (V, E)$  with  $|V| = n$  and  $|E| = m$ , construct a tree  $T$  as follows:

- create a pivot root  $s$  with color  $c_s$ ;
- for each edge  $e_i = uv$  of  $G$ , add to  $T$  a subset of vertices  $E_i = \{u'_i, v'_i, u''_i, v''_i\}$  such that  $u'_i, v'_i$  are children of  $s$ ,  $v''_i$  is a child of  $u'_i$ , and  $u''_i$  is a child of  $v'_i$ ;
- define a distinct color  $c_u$  for each  $u \in V(G)$ , and color all vertices of the form  $u'_i, u''_i$  (for all  $i$ ) with the color  $c_u$ .

Figure 2 shows a graph  $G$  and its associated tree  $T$ .



**Fig. 2.** (a) A graph  $G$ ; (b) tree  $T$  obtained from  $G$

Suppose that  $G$  has a vertex cover  $V'$  of size  $k$ . By construction, the set of vertices not flooded have  $n$  colors. By playing  $n$  moves (using all the  $n$  colors, one for each move), each subset  $E_i$  still contains a vertex not m-connected to  $s$ . Thus we can play these  $n$  moves in the following order: initially, moves played on colors assigned to vertices of  $V'$ , and then other moves. Since every edge in  $G$  contains at least one of its endpoints in  $V'$ , after these  $n$  moves the vertices in  $T$  not m-connected to  $s$  have colors associated with vertices of  $V'$ . Since  $|V'| = k$ , we will need at most  $k$  additional moves, and therefore the flooding will have  $n + k$  moves, as required.

Now assume that  $T$  has a flooding with  $n+k$  moves. Initially,  $T$  contains only the color  $c_s$  and  $n$  other colors forming a set  $C$  of colors. Hence, each color of  $C$  is played at least once. We divide the colors of  $C$  into two groups: the first group is formed by the colors played more than once, and the second group by colors played only once. In order to flood the subset  $E_i$ , the first and the last moves

played on it are moves played on colors of the first group. Hence, without loss of generality, we can assume that the  $n+k$  moves are played in the following order: (a) the first move of all colors in the first group, (b) the moves of the colors in the second group, and (c) the remaining moves of the colors in the first group. Thus, after playing the moves corresponding to (a) and (b) (note that these are  $n$  moves, one for each color), one vertex in each subset  $E_i$  remains unflooded. In addition, vertices not m-connected to  $s$  have  $k$  colors. Since each color in  $T$  represents a distinct vertex in  $G$  and in the construction each  $E_i$  is associated with a distinct edge in  $G$ , these  $k$  colors correspond to a subset of vertices in  $G$  of size  $k$  that are a vertex cover of  $G$ .  $\square$

**Theorem 2.** *Flood-It on trees whose leaves are at distance at most  $d$  from the pivot, for a constant  $d$ , admits a polynomial kernelization (and thus is in FPT) when parameterized by the number of colors  $c$ .*

**Proof.** Let  $T$  be a tree with  $n$  vertices and pivot  $p$ . We show below how to find a polynomial kernel (i.e., a kernel whose size is bounded by a polynomial in  $c$ ) for the problem, in  $O(n)$  time. Apply the following kernelization algorithm:

1. set  $T' = T$ ;
2. contract all children of  $p$  in  $T'$  with color  $c_i$  into a single vertex of color  $c_i$ . Note that this rule can be applied since the contracted vertices will always be flooded by the same move in  $T$ ;
3. recursively repeat the previous step for each non-leaf child of  $p$  in  $T'$ .

After applying the above algorithm, each vertex in  $T'$  has at most  $c$  children. Thus,  $T'$  has at most  $c^d$  vertices and is a polynomial kernel for the problem.  $\square$

**Definition 1.** *Two problems  $\Pi$  and  $\Pi'$  are said to be analogous if there exist linear-time computable functions  $f, g$  such that:*

1.  $\Pi \propto^f \Pi'$  and  $\Pi' \propto^g \Pi$ ;
2. every solution  $s$  for an instance  $I$  of  $\Pi$  implies a solution  $s'$  for  $f(I)$  such that  $\text{size}(s) = \text{size}(s')$ ;
3. every solution  $s'$  for an instance  $I'$  of  $\Pi'$  implies a solution  $s$  for  $g(I')$  such that  $\text{size}(s') = \text{size}(s)$ .

**Definition 2.** *Let  $\Pi$  and  $\Pi'$  be analogous problems. The parameterized problems  $\Pi(k_1, \dots, k_\ell)$  and  $\Pi'(k'_1, \dots, k'_\ell)$  are said to be p-analogous if there exist FPT reductions  $f, g$  and a one-to-one correspondence  $k_i \leftrightarrow k'_i$  such that:*

1.  $\Pi(k_1, \dots, k_\ell) \propto^f \Pi'(k'_1, \dots, k'_\ell)$  and  $\Pi'(k'_1, \dots, k'_\ell) \propto^g \Pi(k_1, \dots, k_\ell)$ ;
2. every solution  $s$  for an instance  $I$  of  $\Pi(k_1, \dots, k_\ell)$  implies a solution  $s'$  for  $f(I)$  such that  $k'_i = \varphi'_i(k_i)$  for some function  $\varphi'_i$  ( $1 \leq i \leq \ell$ );
3. every solution  $s'$  for an instance  $I'$  of  $\Pi'(k'_1, \dots, k'_\ell)$  implies a solution  $s$  for  $g(I')$  such that  $k_i = \varphi_i(k'_i)$  for some function  $\varphi_i$  ( $1 \leq i \leq \ell$ ).

Two easy consequences of the above definitions are: (a) if  $\Pi$  and  $\Pi'$  are analogous problems then  $\Pi$  is in P (is NP-hard) if and only if  $\Pi'$  is in P (is NP-hard); (b) if

$\Pi(k_1, \dots, k_\ell)$  and  $\Pi'(k'_1, \dots, k'_\ell)$  are p-analogous problems then  $\Pi(k_1, \dots, k_\ell)$  is in FTP (admits a polynomial kernel/is W[1]-hard) if and only if  $\Pi'(k'_1, \dots, k'_\ell)$  is in FTP (admits a polynomial kernel/is W[1]-hard).

Lagoutte, Noual, and Thierry used a reduction from the Fixed Alphabet Shortest Common Supersequence problem [16], to prove that Flood-It on trees is NP-hard even when the number of colors is fixed. We show that Flood-It on trees and Restricted Shortest Common Supersequence (RSCS) are analogous problems. RSCS is a variant of SCS - Shortest Common Supersequence [9].

### Shortest Common Supersequence (decision version)

*Instance:* A set of strings  $S = s_1, \dots, s_k$  over an alphabet  $\Sigma$ , an integer  $\lambda$ .

*Question:* Does there exist a string  $s \in \Sigma$  of length at most  $\lambda$  that is a supersequence of each string in  $S$ ?

### Restricted Shortest Common Supersequence (decision version)

*Instance:* A set of  $\rho$ -strings<sup>1</sup>  $R = r_1, \dots, r_k$  over an alphabet  $\Sigma$ , an integer  $\lambda$ .

*Question:* Does there exist a string  $r \in \Sigma$  of length at most  $\lambda$  that is a supersequence of each  $\rho$ -string in  $R$ ?

Let SCS( $|\Sigma_1|, k_1$ ) stand for the SCS problem parameterized by  $|\Sigma_1|$  and  $k_1$  ( $k_1$  is the number of strings). The notation RCSC( $|\Sigma_2|, k_2$ ) is used similarly.

**Theorem 3.** *SCS( $|\Sigma_1|, k_1$ ) is FPT-reducible to RSCS( $|\Sigma_2|, k_2$ ).*

**Proof.** Let  $I$  be an instance of SCS( $|\Sigma_1|, k_1$ ). Create an instance  $I'$  of problem RSCS( $|\Sigma_2|, k_2$ ) as follows: for each string  $s_i$  of  $I$  define a  $\rho$ -string  $r_i$  of  $I'$  by inserting a new symbol  $c_i$  after each symbol of  $s_i$ . After this construction,  $I'$  contains  $k_2 = k_1$   $\rho$ -strings over an alphabet  $\Sigma_2$  such that  $|\Sigma_2| = |\Sigma_1| + k_1$ . At this point, it is easy to see that  $I$  contains a supersequence of length  $\ell$  if and only if  $I'$  contains a supersequence of length  $|s_1| + \dots + |s_{k_1}| + \ell$ .  $\square$

**Theorem 4.** (a) *Flood-It on trees and RSCS are analogous problems.* (b) *Flood-It on trees parameterized by number of colors, number of leaves, and number of moves is p-analogous to RSCS( $|\Sigma|, k, q$ ), where  $q$  is the length of the solution string.*

**Proof.** We prove only item (a). Given an instance  $I$  of RSCS, we create a colored tree  $T$  as follows: (i) each position of a string in  $I$  is converted into a vertex of  $T$ ; (ii) if a position of a string in  $I$  contains a character  $c$  then the corresponding vertex of  $T$  receives color  $c$ ; (iii) an edge is added between two vertices of  $T$  if and only if they represent consecutive positions of the same string; (iv) a pivot vertex  $p$  is created in  $T$  with a new color; (v) an edge  $(p, v)$  is added to  $T$  if  $v$  represents the first position of a string in  $I$ .

After this construction, note that if  $I$  admits a supersequence of size  $\ell$  then  $T$  has a flooding with  $\ell$  moves, obtained by traversing the supersequence and playing color  $c$  in the  $j^{\text{th}}$ -move if character  $c$  is in the  $j^{\text{th}}$ -position of the supersequence. Similarly, given a flooding  $F$  of  $T$  with  $\ell$  moves, we can construct a

---

<sup>1</sup> A  $\rho$ -string is a string with no identical consecutive symbols.

supersequence  $s$  of size  $\ell$  for  $I$ , by just adding character  $c$  in position  $j$  of  $s$  if and only if color  $c$  is played in the  $j^{\text{th}}$ -move of  $F$ .

On the other hand, given an instance  $T$  of Flood-It on trees, we create an instance  $I$  of RSCS as follows: (i) each color in  $T$  is associated with a character of the alphabet  $\Sigma$  over which strings in  $I$  are defined; (ii) for each path  $P$  from the pivot to a leaf in  $T$ , a string  $r(P)$  of  $I$  is created by first contracting each maximal monochromatic subgraph of  $P$  into a single vertex with the same color, and then by adding character  $c$  in the  $j^{\text{th}}$ -position of  $r(P)$  if the vertex in  $P$  at distance  $j > 0$  from the pivot has color  $c$ . If  $T$  has a flooding with  $\ell$  moves then  $I$  admits a supersequence of size  $\ell$ , since, in order to flood a leaf, one needs to flood the path that connects it to the pivot. As previously, if  $I$  admits a supersequence of size  $\ell$  then  $T$  has a flooding with  $\ell$  moves, obtained by traversing the supersequence and playing color  $c$  in the  $j^{\text{th}}$ -move if character  $c$  is in the  $j^{\text{th}}$ -position of the supersequence.  $\square$

**Observation.** By Theorem 2 and Theorem 4(b), the problem RSCS parameterized by  $|\Sigma|$ , where  $\rho$ -strings have length bounded by a constant  $d$ , admits a polynomial kernel, namely, a data structure known as *trie* [11] constructed from the input  $\rho$ -strings. Such a data structure is a prefix tree with at most  $|\Sigma|^d$  nodes.

**Corollary 5.** *Flood-It on paths with arbitrary pivot is analogous to RSCS for  $k \leq 2$ .*  $\square$

By Theorem 4, results valid for RSCS can be inherited by Flood-It on trees:

**Corollary 6.** *Flood-It on trees can be solvable in polynomial time when restricted to trees with constant number of leaves.*

**Proof.** Follows from Theorem 4(a) and the analogous result in [17]: SCS (and thus RSCS) is solvable in polynomial time for a constant number of strings.  $\square$

**Corollary 7.** *Flood-It on trees is W[1]-hard when parameterized by the number of leaves and the number of colors.*

**Proof.** Follows from Theorems 3, Theorem 4(b), and the analogous result in [22]: SCS( $|\Sigma_1|, k_1$ ) is W[1]-hard.  $\square$

**Definition 3.** A colored rooted tree is a pc-tree (phylogenetic colored tree) if no color occurs more than once in any path from the root to a leaf.

**Corollary 8.** *Flood-It on trees remains NP-hard even when restricted to pc-trees with pivot root.*

**Proof.** Follows from Theorem 4(a) and the analogous result in [9]: SCS (and also RSCS) is NP-hard even for strings where no symbol occurs more than once.  $\square$

**Corollary 9.** *Flood-It on pc-trees with pivot root is W[1]-hard when parameterized by the number of leaves.*

**Proof.** Follows from Theorem 4(b) and the analogous result in [9]: SCS (and also RSCS) restricted to strings where no symbol occurs more than once is W[1]-hard when parameterized by the number of strings.  $\square$

**Corollary 10.** *Flood-It on pc-trees with pivot root and  $k$  leaves is in FPT when it is asked whether there is a flooding with at most  $c + r$  moves, where  $c$  is the number of colors and the pair  $(k, r)$  is the parameter.*

**Proof.** Follows from Theorem 4(b) and the analogous result in [9]: SCS (and thus RSCS) is in FPT when, in the  $k$  input strings, each symbol occurs at most once, and the question is whether there is a common supersequence of size bounded by  $|\Sigma| + r$ , where the parameter is the pair  $(k, r)$ .  $\square$

**Definition 4.** *A pc-tree  $T$  is a cpc-tree (complete pc-tree) if each color occurs exactly once in any path from the root to a leaf.*

**Corollary 11.** *Flood-It on cpc-trees with pivot root is in FPT when it is asked whether there is a flooding with at most  $c + r$  moves, where  $c$  is the number of colors and  $r$  is the parameter.*

**Proof.** Follows from Theorem 4(b) and the analogous result in [9]: SCS (and thus RSCS) is in FPT when every symbol occurs exactly once in each string and the question is whether there is a common supersequence of size bounded by  $|\Sigma| + r$ , where  $r$  is the parameter.  $\square$

## 2.1 Multi-Flood-It on Trees

In this subsection we deal with a new variant of Flood-It, *Multi-Flood-It*, where each move is played on a set of fixed pivots. We assume that, before a move  $m$ , all the pivots have the same color. The effect of playing a color  $c$  in move  $m$  is assigning  $c$  to the pivots and to every vertex m-connected to some pivot immediately before playing  $m$ .

We consider Multi-Flood-It played on trees where the pivots are precisely the leaves, called *Multi-Flood-It on trees* for short.

**Theorem 12.** *Flood-It on trees with pivot root is reducible to Multi-Flood-It on trees.*

**Proof.** Let  $T$  be an instance of Flood-it on trees with pivot root. We create an instance  $T'$  of Multi-Flood-It on trees with leaf pivots as follows:

- For each path  $p_i$  in  $T$  from the root to a leaf  $l_i$  do:
  1. create two copies  $p_i^1$  and  $p_i^2$  of  $p_i$ , keeping the same colors of the vertices in  $p_i$ ;
  2. let  $r_i^j$  and  $l_i^j$  denote, respectively, the copy of the root of  $T$  in  $p_i^j$  and the copy of  $l_i$  in  $p_i^j$ ,  $j = 1, 2$ ;
  3. add edge  $(l_i^1, l_i^2)$ ;
- Contract the vertices  $r_i^1$  (for all  $i$ ) into a single vertex  $r$ ;

- Create a new vertex  $u$  with the same color as  $r$ , and add edge  $(u, r)$ .

At this point, it is easy to see that  $T$  has a flooding of size  $k$  using pivot root if and only if  $T'$  has a flooding of size  $k$  using the leaf pivots.  $\square$

**Corollary 13.** *Multi-Flood-It on trees is NP-hard.*

**Proof.** Follows from Theorems 1 and 12.  $\square$

**Corollary 14.** *Multi-Flood-It on trees is W[1]-hard when parameterized by the number of leaves and the number of colors.*

**Proof.** From Theorem 12 it is easy to see that Flood-It on trees and Multi-Flood-It on trees, both parameterized by the number of leaves and number of colors, are p-analogous. Thus, by Corollary 7, the result follows.  $\square$

**Theorem 15.** *Multi-Flood-It on trees is in FPT when the number of leaves is  $k$  and it is asked whether there is a flooding with at most  $c + r$  moves, where  $c$  is the number of colors and the pair  $(k, r)$  is the parameter.*

**Proof.** Suppose the color  $c_a$  is chosen for a move of the game. For each path  $p_i$  from a leaf  $l_i$  to the root, one of the following statements must be true: (1) The color  $c_a$  is the first color of  $p_i$  (different of the pivot color) and does not otherwise occur in the current  $p_i$ ; (2) The color  $c_a$  does not occur in the current  $p_i$ ; (3) The color  $c_a$  occurs in the current  $p_i$ , but is not the first color. If for a move of the game only (1) and (2) occur, we call this a *good move*. A move that is not good is *bad*. Our algorithm is based on the following claims:

*Claim 1.* If at least  $r$  bad moves are played then  $T$  has a flooding with at least  $c + r$  moves.

*Claim 2.* For any yes-instance of the problem, there is a flooding with at most  $r$  bad moves.

As in [9], we can describe an FPT-algorithm based on the method of search trees [7]. By Claim 2, if the answer is “yes” then there is a game that completes with no more than  $r$  bad moves. The algorithm is as follows:

- (0) The root node of the search tree is labeled with the given input.
- (1) A node of the search tree is expanded by making a sequence of good moves (arbitrarily) until no good move is possible. For each possible nontrivial bad move (i.e., one that floods at least one vertex), create a child node labeled with the set of sequences that result after this bad move.
- (2) If a node is labeled by the set of empty sequences, then answer “yes”.
- (3) If a node has depth  $r$  in the search tree, then do not expand any further.

The correctness of the algorithm follows from Claims 1 and 2, and the fact that the sequence of good moves in step (1) can be made in any order without increasing the number of moves. The running time of the algorithm is bounded by  $O(k^r n)$ .  $\square$

It is easy to see that we can extend this FPT-algorithm to Flood-It on trees with pivot root.

### 3 Free-Flood-It on Trees

**Definition 5.** Let  $G$  be a graph,  $v \in V(G)$ , and  $\ell$  a positive integer. The graph  $\psi(G, v, \ell)$  is constructed as follows: (i) create  $\ell$  disjoint copies  $G_1, G_2, \dots, G_\ell$  of  $G$ ; (ii) contract the copies  $v_1, v_2, \dots, v_\ell$  of  $v$  into a single vertex  $v^*$ .

**Definition 6.** Let  $\mathcal{F}$  be a class of graphs. Then:

$$\psi(\mathcal{F}) = \{G \mid G = \psi(G', v, \ell) \text{ for some triple } (G' \in \mathcal{F}, v \in V(G'), \ell > 0)\}.$$

**Definition 7.** A class  $\mathcal{F}$  of graphs is closed under operator  $\psi$  if  $\psi(\mathcal{F}) \subseteq \mathcal{F}$ .

Examples of classes closed under  $\psi$  are chordal graphs and bipartite graphs.

**Theorem 16.** Flood-It played on  $\mathcal{F}$  is reducible in polynomial time to Free-Flood-It played on  $\psi(\mathcal{F})$ .

**Proof.** Let  $G$  be an instance of Flood-It on  $\mathcal{F}$  (with pivot  $p$ ). Assume  $|V(G)| = n$ . We create an instance for Free-Flood-It on  $\psi(\mathcal{F})$  by constructing the graph  $G' = \psi(G, p, n)$  and coloring a vertex  $w_i$  in copy  $G_i$  with the same initial color of its corresponding vertex  $w \in V(G)$ . Now we show that there is a flooding for  $G$  with at most  $k$  moves if and only if there is a free-flooding for  $G'$  with at most  $k$  moves, as follows. First, note that every flooding  $F$  for  $G$  implies a free-flooding  $F'$  for  $G'$  with the same number of moves as  $F$ , by simply using  $p^*$  as the pivot of all moves in  $F'$  and repeating the same sequence of colors played in  $F$ . Conversely, if there is a flooding  $F'$  for  $G'$  with at most  $k$  moves, then: (i) If on every subgraph  $G_i$  ( $1 \leq i \leq n$ ) of  $G'$  a move is played that does not change the color of  $p^*$  then  $k \geq |F'| \geq n$ ; in this case, it is easy to see that there is a flooding for  $G$  with at most  $k$  moves, since  $|V(G)| = n$  and thus  $n - 1$  moves suffice to flood  $G$ . (ii) If there is a subgraph  $G_i$  such that every move played on  $G_i$  changes the color of  $p^*$  then, without loss of generality, the same sequence of colors played in such moves can be used to flood  $G$ , using  $p$  as a fixed pivot.  $\square$

**Corollary 17.** Let  $\mathcal{F}$  be a class of graphs closed under  $\psi$ . Then Flood-It played on  $\mathcal{F}$  is reducible in polynomial time to Free-Flood-It played on  $\mathcal{F}$ .

NP-hardness results valid for Flood-It can be inherited by Free-Flood-It:

**Corollary 18.** Free-Flood-It remains NP-hard on trees where each leaf is at a distance at most four of any other vertex.

**Proof.** Follows from Theorem 16 and Theorem 1.  $\square$

**Corollary 19.** Free-Flood-It on pc-trees is NP-hard.

**Proof.** Follows from Corollary 17 and Corollary 8.  $\square$

**Theorem 20.** In Free-Flood-It on pc-trees, there always exists an optimal free-flooding which is a flooding with pivot root.

**Proof.** Let  $T$  be a pc-tree with root  $p$ . Let  $h(T)$  denote the height of  $T$ , and let  $C = \{c_1, c_2, \dots, c_k\}$  be the set of colors assigned to the leaves at level  $h = h(T)$  (the root is at level 0). We use induction on  $h(T)$ . The result is clearly valid when  $h(T) = 1$ , since the sequence of moves  $(p, c_1), (p, c_2), \dots, (p, c_k)$  is an optimal free-flooding of  $T$  which is a flooding with pivot  $p$ . Now assume that the result is valid for all pc-trees with height at most  $h - 1$ . By induction, there exists an optimal free-flooding  $F'$  of the subtree  $T'$  obtained from  $T$  by removing all the leaves at level  $h$  in  $T$ , such that  $F'$  is a flooding with pivot  $p$ . Consider the flooding  $F$  of  $T$  by appending to  $F'$  the sequence of moves  $(p, c_1), (p, c_1), \dots, (p, c_k)$ . Then  $F$  is an optimal free-flooding of  $T$  which is a flooding with pivot  $p$ .  $\square$

The above theorem implies that Flood-It on pc-trees and Free-Flood-It on pc-trees are analogous, and parameterized versions of these problems are p-analogous. Thus:

**Corollary 21.** *Free-Flood-It on pc-trees is W[1]-hard when parameterized by the number of leaves.*

**Proof.** Follows from Theorem 20 and Corollary 9.  $\square$

**Corollary 22.** *Free-Flood-It on pc-trees with pivot root and  $k$  leaves is in FPT when it is asked whether there is a free-flooding with at most  $c + r$  moves, where  $c$  is the number of colors and the pair  $(k, r)$  is the parameter.*

**Proof.** Follows from Theorem 20 and Corollary 10.  $\square$

**Corollary 23.** *Free-Flood-It on cpc-trees with pivot root is in FPT when it is asked whether there is a free-flooding with at most  $c + r$  moves, where  $c$  is the number of colors and  $r$  is the parameter.*

**Proof.** Follows from Theorem 20 and Corollary 11.  $\square$

**Acknowledgements.** We are very grateful to Michael Fellows for his insightful ideas on the problems dealt with in this work.

## References

- Arthur, D., Clifford, R., Jalsenius, M., Montanaro, A., Sach, B.: The Complexity of Flood Filling Games. In: Boldi, P., Gargano, L. (eds.) FUN 2010. LNCS, vol. 6099, pp. 307–318. Springer, Heidelberg (2010)
- Aschwanden, C.: Spatial Simulation Model for Infectious Viral Disease with Focus on SARS and the Common Flu. In: Proceedings of the 37th Annual Hawaii International Conference on System Sciences. IEEE Computer Society, Washington, DC (2004)
- Barone, P., Bonizzoni, P., Vedova, G.D., Mauri, G.: An Approximation Algorithm for the Shortest Common Supersequence Problem: An Experimental Analysis. In: ACM Symposium on Applied Computing, pp. 56–60 (2001)
- Bodlaender, H.L., Fellows, M.R., Hallett, M.T., Wareham, T., Warnow, T.: The Hardness of Perfect Phylogeny, Feasible Register Assignment and other Problems on Thin Colored Graphs. Theoretical Computer Science 244, 167–188 (2000)

5. Chor, B., Fellows, M.R., Ragan, M.A., Razgon, I., Rosamond, F.A., Snir, S.: Connected Coloring Completion for General Graphs: Algorithms and Complexity. In: Lin, G. (ed.) COCOON 2007. LNCS, vol. 4598, pp. 75–85. Springer, Heidelberg (2007)
6. Clifford, R., Jalsenius, M., Montanaro, A., Sach, B.: The Complexity of Flood-Filling Games. *Theory of Computing Systems* 50(1), 72–92 (2012)
7. Downey, R.G., Fellows, M.R.: Parametrized Computational Feasibility. In: Clote, P., Remmel, J. (eds.) *Feasible Mathematics II*, pp. 219–244. Birkhauser, Boston (1995)
8. Fellows, M.R., Fertin, G., Hermelin, D., Vialette, S.: Sharp Tractability Borderlines for Finding Connected Motifs in Vertex-Colored Graphs. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 340–351. Springer, Heidelberg (2007)
9. Fellows, M.R., Hallett, M.T., Stege, U.: Analogs & Duals of the MAST problem for Sequences & Trees. *Journal of Algorithms* 49(1), 192–216 (2003)
10. Fellows, M.R., Hallett, M.T., Wareham, H.T.: DNA Physical Mapping: Three Ways Difficult. In: Lengauer, T. (ed.) ESA 1993. LNCS, vol. 726, pp. 157–168. Springer, Heidelberg (1993)
11. Fredkin, E.: Trie Memory. *Communications of the ACM* 3, 490–499 (1960)
12. Golumbic, M., Kaplan, H., Shamir, R.: On the Complexity of DNA Physical Mapping. *Advances in Applied Mathematics* 15, 251–261 (1994)
13. Gusfield, D.: Efficient Algorithms for Inferring Evolutionary Trees. *Networks* 21, 19–28 (1981)
14. Hallett, M.T.: An Integrated Complexity Analysis of Problems from Computational Biology. PhD thesis, University of Victoria (1996)
15. Lacroix, V., Fernandes, C.G., Sagot, M.F.: Motif Search in Graphs: Application to Metabolic Networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 3(4), 360–368 (2006)
16. Lagoutte, A., Noual, M., Thierry, E.: Flooding Games on Graphs, HAL: hal-00653714 (December 2011)
17. Maier, D.: The Complexity of Some Problems on Subsequences and Supersequences. *Journal of the ACM* 25(2), 322–336 (1978)
18. McMorris, F.R., Warnow, T.J., Wimer, T.: Triangulating Vertex-Colored Graphs. *SIAM Journal on Discrete Mathematics* 7(2), 296–306 (1994)
19. Meeks, K., Scott, A.: The Complexity of Flood-Filling Games on Graphs. *Discrete Applied Mathematics* 160, 959–969 (2012)
20. Meeks, K., Scott, A.: The Complexity of Free-Flood-It on  $2 \times n$  Boards, arXiv:1101.5518v1 [cs.DS] (January 2011)
21. Moran, S., Snir, S.: Convex Recolorings of Strings and Trees: Definitions, Hardness Results and Algorithms. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 218–232. Springer, Heidelberg (2005)
22. Pietrzak, K.: On the Parameterized Complexity of the Fixed Alphabet Shortest Common Supersequence and Longest Common Subsequence Problems. *Journal of Computer and System Sciences* 67(4), 757–771 (2003)
23. Rahmann, S.: The Shortest Common Supersequence Problem in a Microarray Production Setting. *Bioinformatics* 19(suppl. 2), ii156–ii161 (2003)
24. Sim, J., Park, K.: The Consensus String Problem for a Metric is NP-complete. *Journal of Discrete Algorithms* 1(1), 111–117 (2003)

# Parameterized Approximability of Maximizing the Spread of Influence in Networks

Cristina Bazgan<sup>1,3</sup>, Morgan Chopin<sup>1</sup>, André Nichterlein<sup>2</sup>, and Florian Sikora<sup>1</sup>

<sup>1</sup> PSL, Université Paris-Dauphine, LAMSADE UMR CNRS 7243, France

{bazgan,chopin,florian.sikora}@lamsade.dauphine.fr

<sup>2</sup> Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany

andre.nichterlein@tu-berlin.de

<sup>3</sup> Institut Universitaire de France

**Abstract.** In this paper, we consider the problem of maximizing the spread of influence through a social network. Here, we are given a graph  $G = (V, E)$ , a positive integer  $k$  and a threshold value  $\text{thr}(v)$  attached to each vertex  $v \in V$ . The objective is then to find a subset of  $k$  vertices to “activate” such that the number of activated vertices at the end of a propagation process is maximum. A vertex  $v$  gets activated if at least  $\text{thr}(v)$  of its neighbors are. We show that this problem is strongly inapproximable in fpt-time with respect to (w.r.t.) parameter  $k$  even for very restrictive thresholds. For unanimity thresholds, we prove that the problem is inapproximable in polynomial time and the decision version is W[1]-hard w.r.t. parameter  $k$ . On the positive side, it becomes  $r(n)$ -approximable in fpt-time w.r.t. parameter  $k$  for any strictly increasing function  $r$ . Moreover, we give an fpt-time algorithm to solve the decision version for bounded degree graphs.

## 1 Introduction

Optimization problems that involve a diffusion process in a graph are well studied [16,12,7,1,11,6,2,17]. Such problems share the common property that, according to a specified *propagation rule*, a chosen subset of vertices activates all or a fixed fraction of the vertices, where initially all but the chosen vertices are inactive. Such optimization problems model the spread of influence or information in social networks via word-of-mouth recommendations, of diseases in populations, or of faults in distributed computing [16,12,11]. One representative problem that appears in this context is the *influence maximization* problem introduced by Kempe *et al.* [12]. Given a directed graph, the task is to choose a vertex subset of size at most a fixed number such that the number of activated vertices at the end of the propagation process is maximized. The authors show that the problem is polynomial-time  $(\frac{e}{e-1} + \varepsilon)$ -approximable for any  $\varepsilon > 0$  under some stochastic propagation models, but NP-hard to approximate within a ratio of  $n^{1-\varepsilon}$  for any  $\varepsilon > 0$  for general propagation rules.

In this paper, we use the following deterministic propagation model. We are given an undirected graph, a threshold value  $\text{thr}(v)$  associated to each vertex

$v$ , and the following propagation rule: a vertex becomes active if at least  $\text{thr}(v)$  many neighbors of  $v$  are active. The propagation process proceeds in several rounds and stops when no further vertex becomes active. Given this model, finding and activating a minimum-size vertex subset such that all or a fixed fraction of the vertices become active is known as the *minimum target set selection* (MinTSS) problem introduced by Chen [7]. It has been shown NP-hard even for bipartite graphs of bounded degree when all thresholds are at most two [7]. Moreover, the problem was surprisingly shown to be hard to approximate within a ratio  $O(2^{\log^{1-\varepsilon} n})$  for any  $\varepsilon > 0$ , even for constant degree graphs with thresholds at most two and for general graphs when the threshold of each vertex is half its degree (called *majority* thresholds) [7]. If the threshold of each vertex equals its degree (*unanimity* thresholds), then the problem is polynomial-time equivalent to the vertex cover problem [7] and, thus, admits a 2-approximation and is hard to approximate with a ratio better than 1.36 [9]. Concerning the parameterized complexity, the problem is shown to be W[2]-hard with respect to (w.r.t.) the solution size, even on bipartite graphs of diameter four with majority thresholds or thresholds at most two [14]. Furthermore, it is W[1]-hard w.r.t. each of the parameters “treewidth”, “cluster vertex deletion number”, and “pathwidth” [2,8]. On the positive side, the problem becomes fixed-parameter tractable w.r.t. each of the single parameters “vertex cover number”, “feedback edge set size”, and “bandwidth” [14,8]. If the input graph is complete, or has a bounded treewidth and bounded thresholds then the problem is polynomial-time solvable [14,2].

Here, we study the complementary problem of MinTSS, called *maximum  $k$ -influence* (Max $k$ Inf) where the task is to maximize the number of activated vertices instead of minimizing the target set size. Since both optimization problems have the same decision version, the parameterized as well as NP-hardness results directly transfer from MinTSS to Max $k$ Inf. We show that also Max $k$ Inf is hard to approximate and, confronted with the computational hardness, we study the parameterized approximability of Max $k$ Inf.

*Our Results.* Concerning the approximability of the problem, there are two possibilities of measuring the value of a solution: counting the vertices activated by the propagation process including or excluding the initially chosen vertices (denoted by MAX CLOSED  $k$ -INFLUENCE and MAX OPEN  $k$ -INFLUENCE, respectively). Observe that whether or not counting the chosen vertices might change the approximation factor. In this paper, we consider both cases and our approximability results are summarized in Table 1.

While MinTSS is both constant-approximable in polynomial time and fixed-parameter tractable for the unanimity case, this does not hold anymore for our problem. Indeed, we prove that, in this case, MAX CLOSED  $k$ -INFLUENCE (resp. MAX OPEN  $k$ -INFLUENCE) is strongly inapproximable in polynomial-time and the decision version, denoted by  $(k, \ell)$ -INFLUENCE, is W[1]-hard w.r.t. the combined parameter  $(k, \ell)$ . However, we show that MAX CLOSED  $k$ -INFLUENCE (resp. MAX OPEN  $k$ -INFLUENCE) becomes approximable if we are allowed to use fpt-time and  $(k, \ell)$ -INFLUENCE gets fixed-parameter tractable w.r.t combined parameter  $(k, \Delta)$ , where  $\Delta$  is the maximum degree of the input graph.

**Table 1.** Table of the approximation results for MAX OPEN  $k$ -INFLUENCE and MAX CLOSED  $k$ -INFLUENCE

| Thresholds | Bounds | MAX OPEN $k$ -INFLUENCE                                |                                                        | MAX CLOSED $k$ -INFLUENCE                              |                                                                |
|------------|--------|--------------------------------------------------------|--------------------------------------------------------|--------------------------------------------------------|----------------------------------------------------------------|
|            |        | poly-time                                              | fpt-time                                               | poly-time                                              | fpt-time                                                       |
| General    | Upper  | $n$                                                    | $n$                                                    | $n$                                                    | $n$                                                            |
|            | Lower  | $n^{1-\varepsilon}, \forall \varepsilon > 0$                   |
| Constant   | Upper  | $n$                                                    | $n$                                                    | $n$                                                    | $n$                                                            |
|            | Lower  | $n^{\frac{1}{2}-\varepsilon}, \forall \varepsilon > 0$ [Th. 2] |
| Majority   | Upper  | $n$                                                    | $n$                                                    | $n$                                                    | $n$                                                            |
|            | Lower  | $n^{1-\varepsilon}, \forall \varepsilon > 0$           | $n^{1-\varepsilon}, \forall \varepsilon > 0$           | $n^{1-\varepsilon}, \forall \varepsilon > 0$           | $n^{1-\varepsilon}, \forall \varepsilon > 0$ [Th. 1]           |
| Unanimity  | Upper  | $2^k$ [Th. 5]                                          | $r(n), \forall r$ [Th. 2]                              | $2^k$                                                  | $r(n), \forall r$                                              |
|            | Lower  | $n^{1-\varepsilon}, \forall \varepsilon > 0$ [Th. 4]   | ?                                                      | $1 + \varepsilon$ [Th. 7]                              | ?                                                              |

Our paper is organized as follows. In Section 2, after introducing some preliminaries, we establish some basic lemmas. In Section 3 we study MAX OPEN  $k$ -INFLUENCE and MAX CLOSED  $k$ -INFLUENCE with majority thresholds and thresholds at most two. In Section 4 we study the case of unanimity thresholds in general graphs and in bounded degree graphs. Conclusions are provided in Section 5. Due to space limitation, some proofs are deferred to a full version.

## 2 Preliminaries and Basic Observations

In this section, we provide basic backgrounds and notation used throughout this paper, give the statements of the studied problems, and establish some lemmas.

*Graph Terminology.* Let  $G = (V, E)$  be an *undirected graph*. For a subset  $S \subseteq V$ ,  $G[S]$  is the subgraph induced by  $S$ . The *open neighborhood* of a vertex  $v \in V$ , denoted by  $N(v)$ , is the set of all neighbors of  $v$ . The *closed neighborhood* of a vertex  $v$ , denoted  $N[v]$ , is the set  $N(v) \cup \{v\}$ . Furthermore, for a vertex set  $V' \subset V$  we set  $N(V') = \bigcup_{v \in V'} N(v)$  and  $N[V'] = \bigcup_{v \in V'} N[v]$ . The set  $N^k[v]$ , called the  $k$ -neighborhood of  $v$ , denotes the set of vertices which are at distance at most  $k$  from  $v$  (thus  $N^1[v] = N[v]$ ). The *degree* of a vertex  $v$  is denoted by  $\deg_G(v)$  and the *maximum degree* of the graph  $G$  is denoted by  $\Delta_G$ . We skip the subscript if  $G$  is clear from the context. Two vertices are *twins* if they have the same neighborhood. They are called *true twins* if they are moreover neighbors, *false twins* otherwise.

*Cardinality Constrained Problem.* The problems studied in this paper are cardinality constrained. We use the notations and definitions from Cai [4]. A cardinality constrained optimization problem is a quadruple  $A = (\mathcal{B}, \Phi, k, obj)$ , where  $\mathcal{B}$  is a finite set called solution base,  $\Phi : 2^{\mathcal{B}} \rightarrow \{0, 1, 2, \dots\} \cup \{-\infty, +\infty\}$  an objective function,  $k$  a non-negative integer and  $obj \in \{\min, \max\}$ . The goal is then to find a solution  $S \subseteq \mathcal{B}$  of cardinality  $k$  so as to maximize (or minimize) the objective value  $\Phi(S)$ . If  $S$  is not a feasible solution we set  $\Phi(S) = -\infty$  if  $obj = \max$  and  $\Phi(S) = +\infty$  otherwise.

*Parameterized Complexity.* A parameterized problem  $(I, k)$  is said *fixed-parameter tractable* (or in the class FPT) w.r.t. parameter  $k$  if it can be solved in  $f(k) \cdot |I|^c$  time, where  $f$  is any computable function and  $c$  is a constant (one can see [10,15]). The parameterized complexity hierarchy is composed of the classes  $\text{FPT} \subseteq \text{W}[1] \subseteq \text{W}[2] \subseteq \dots \subseteq \text{W[P]}$ . A W[1]-hard problem is not fixed-parameter tractable (unless FPT = W[1]) and one can prove W[1]-hardness by means of a *parameterized reduction* from a W[1]-hard problem. This is a mapping of an instance  $(I, k)$  of a problem  $A_1$  in  $g(k) \cdot |I|^{O(1)}$  time (for any computable  $g$ ) into an instance  $(I', k')$  for  $A_2$  such that  $(I, k) \in A_1 \Leftrightarrow (I', k') \in A_2$  and  $k' \leq h(k)$  for some  $h$ .

*Approximation.* Given an optimization problem  $Q$  and an instance  $I$  of this problem, we denote by  $|I|$  the size of  $I$ , by  $\text{opt}_Q(I)$  the optimum value of  $I$  and by  $\text{val}(I, S)$  the value of a feasible solution  $S$  of  $I$ . The *performance ratio* of  $S$  (or *approximation factor*) is  $r(I, S) = \max \left\{ \frac{\text{val}(I, S)}{\text{opt}_Q(I)}, \frac{\text{opt}_Q(I)}{\text{val}(I, S)} \right\}$ . The *error* of  $S$ ,  $\varepsilon(I, S)$ , is defined by  $\varepsilon(I, S) = r(I, S) - 1$ . For a function  $f$  (resp. a constant  $c > 1$ ), an algorithm is a  $f(n)$ -approximation (resp. a  $c$ -approximation) if for any instance  $I$  of  $Q$  it returns a solution  $S$  such that  $r(I, S) \leq f(n)$  (resp.  $r(I, S) \leq c$ ). An optimization problem is polynomial-time *constant approximable* (resp. has a *polynomial-time approximation scheme*) if, for some constant  $c > 1$  (resp. every constant  $\varepsilon > 0$ ), there exists a polynomial-time  $c$ -approximation (resp.  $(1 + \varepsilon)$ -approximation) for it. An optimization problem is  *$f(n)$ -approximable in fpt-time w.r.t. parameter  $k$*  if there exists an  $f(n)$ -approximation running in time  $g(k) \cdot |I|^c$ , where  $k$  is a positive integer depending on  $I$ ,  $g$  is any computable function and  $c$  is a constant [13]. For a cardinality constrained problem a possible choice for the parameter is the cardinality of the solutions.

*Problems definition.* Let  $G = (V, E)$  be an undirected graph and a threshold function  $\text{thr} : V \rightarrow \mathbb{N}$ . In this paper, we consider majority thresholds *i.e.*  $\text{thr}(v) = \lceil \frac{\deg(v)}{2} \rceil$  for each  $v \in V$ , unanimity thresholds *i.e.*  $\text{thr}(v) = \deg(v)$  for each  $v \in V$ , and constant thresholds *i.e.*  $\text{thr}(v) \leq c$  for each  $v \in V$  and some constant  $c > 1$ . Initially, all vertices are not activate and we select a subset  $S \subseteq V$  of  $k$  vertices. The propagation unfolds in discrete steps. At time step 0, only the vertices in  $S$  are activated. At time step  $t + 1$ , a vertex  $v$  is activated if and only if the number of its activated neighbors at time  $t$  is at least  $\text{thr}(v)$ . We apply the rule iteratively until no more activations are possible. Given that  $S$  is the set of initially activated vertices, *closed activated vertices*, denoted by  $\sigma[S]$  is the set of all activated vertices at the end of the propagation process and *closed activated vertices*, denoted by  $\sigma(S)$ , is the set  $\sigma[S] \cup S$ . The optimization problems we consider are then defined as follows.

MAX OPEN  $k$ -INFLUENCE

**Input:** A graph  $G = (V, E)$ , a threshold function  $\text{thr} : V \rightarrow \mathbb{N}$ , and an integer  $k$ .

**Output:** A subset  $S \subseteq V$ ,  $|S| \leq k$  such that  $|\sigma(S)|$  is maximum.

Similarly, the MAX CLOSED  $k$ -INFLUENCE problem asks for a set  $S$  such that  $|\sigma[S]|$  is maximum. The corresponding decision version  $(k, \ell)$ -INFLUENCE is also studied. Notice that, in this case, considering either the open or closed activated vertices is equivalent.

**$(k, \ell)$ -INFLUENCE**

**Input:** A graph  $G = (V, E)$ , a threshold function  $\text{thr} : V \rightarrow \mathbb{N}$ , and two integers  $k$  and  $\ell$ .

**Output:** Is there a subset  $S \subseteq V$ ,  $|S| \leq k$  such that  $|\sigma(S)| \geq \ell$  ?

*Basic results.* In the following, we state and prove some lemmas that will be used later in the paper.

**Lemma 1.** *Let  $r$  be any computable function. If MAX OPEN  $k$ -INFLUENCE is  $r(n)$ -approximable then MAX CLOSED  $k$ -INFLUENCE is also  $r(n)$ -approximable where  $n$  is the input size.*

*Proof.* Let  $A$  be an  $r(n)$ -approximation algorithm for MAX OPEN  $k$ -INFLUENCE. Let  $I$  be an instance of MAX CLOSED  $k$ -INFLUENCE and  $\text{opt}(I)$  its optimum value. When we apply  $A$  on  $I$  it returns a solution  $S$  such that  $|\sigma(S)| \geq \frac{\text{opt}(I)-k}{r(n)}$  and then  $|\sigma[S]| = k + |\sigma(S)| \geq \frac{\text{opt}(I)}{r(n)}$ .  $\square$

**Lemma 2.** *Let  $I$  be an instance of a cardinality constrained optimization problem  $A = (\mathcal{B}, \Phi, k, \text{obj})$ . If  $A$  is  $r_1(k)$ -approximable in fpt-time w.r.t. parameter  $k$  for some strictly increasing function  $r_1$  then it is also  $r_2(|\mathcal{B}|)$ -approximable in fpt-time w.r.t. parameter  $k$  for any strictly increasing function  $r_2$ .*

*Proof.* Let  $r_1^{-1}$  and  $r_2^{-1}$  be the inverse functions of  $r_1$  and  $r_2$ , respectively. We distinguish the following two cases.

**Case 1:**  $k \leq r_1^{-1}(r_2(|\mathcal{B}|))$ . In this case, we apply the  $r_1(k)$ -approximation algorithm and directly get the  $r_2(|\mathcal{B}|)$ -approximation in time  $f(k) \cdot |\mathcal{B}|^{O(1)}$  for some computable function  $f$ .

**Case 2:**  $k > r_1^{-1}(r_2(|\mathcal{B}|))$ . We then have  $|\mathcal{B}| < r_2^{-1}(r_1(k))$ . In this case, we solve the problem exactly by brute-force. If  $\text{obj} = \max$  (resp.  $\text{obj} = \min$ ) then try all possible subset  $S \subseteq \mathcal{B}$  of size  $k$  and take the one that maximizes (resp. minimizes) the objective value  $\Phi(S)$ . The running time is then  $O(|\mathcal{B}|^k) = O(r_2^{-1}(r_1(k))^k)$ .

The overall running time is  $O(\max\{r_2^{-1}(r_1(k))^k, f(k) \cdot |\mathcal{B}|^{O(1)}\})$ , that is, fpt-time.  $\square$

It is worth pointing out that a problem which is proven inapproximable in fpt-time obviously implies that it is not approximable in polynomial time with the same ratio. Therefore, fpt-time inapproximability can be considered as a “stronger” result than polynomial-time inapproximability.

### 3 Parameterized Inapproximability

In this section, we consider the parameterized approximability of both MAX CLOSED  $k$ -INFLUENCE and MAX OPEN  $k$ -INFLUENCE. We show that these problems are W[2]-hard to approximate within  $n^{1-\varepsilon}$  and  $n^{\frac{1}{2}-\varepsilon}$  for any  $\varepsilon \in (0, 1)$  for majority thresholds and thresholds at most two, respectively. To do so, we use the following construction from DOMINATING SET as the starting point. The DOMINATING SET problem asks, given an undirected graph  $G = (V, E)$  and an integer  $k$ , whether there is a vertex subset  $S \subseteq V$ ,  $|S| \leq k$ , such that  $N[S] = V$ .

*Basic Reduction.* Given an instance  $(G = (V, E), k)$  of DOMINATING SET we construct a bipartite graph  $G' = (V', E')$  as follows. For each vertex  $v \in V$  we add two vertices  $v^t$  and  $v^b$  ( $t$  and  $b$  respectively standing for *top* and *bottom*) to  $V'$ . For each edge  $\{u, v\} \in E$  add the edge  $\{v^t, u^b\}$ . Finally, set  $\text{thr}(v^t) = \deg_{G'}(v^t)$  and  $\text{thr}(v^b) = 1$  for every top vertex  $v^t$  and every bottom vertex  $v^b$ , respectively. Clearly, the construction can be computed in polynomial time and, furthermore, it has the following property.

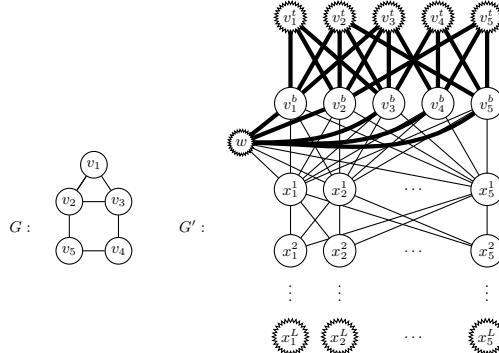
**Lemma 3.** *Let  $G' = (V', E')$  be the graph obtained from a graph  $G$  using the above construction. Then  $G$  admits a dominating set of size  $k$  if and only if  $G'$  admits a subset  $S' \subseteq V'$  of size  $k$  such that  $\sigma[S'] = V'$ .*

*Inapproximability Results.* We are now ready to prove the main results of this section.

**Theorem 1.** *For any  $\varepsilon \in (0, 1)$ , MAX CLOSED  $k$ -INFLUENCE and MAX OPEN  $k$ -INFLUENCE with majority thresholds cannot be approximated within  $n^{1-\varepsilon}$  in fpt-time w.r.t. parameter  $k$  even on bipartite graphs, unless FPT = W[2].*

*Proof.* By Lemma 1, it suffices to show the result for MAX CLOSED  $k$ -INFLUENCE. We construct a polynomial-time reduction from DOMINATING SET to MAX CLOSED  $(k+1)$ -INFLUENCE with majority. In this reduction, we will make use of the  $\ell$ -edge gadget, for some integer  $\ell$ . An  $\ell$ -edge between two vertices  $u$  and  $v$  consists of  $\ell$  vertices of threshold one adjacent to both  $u$  and  $v$ .

Given an instance  $I = (G = (V, E), k)$  of DOMINATING SET with  $n = |V|$ ,  $m = |E|$ , we define an instance  $I'$  of MAX CLOSED  $(k+1)$ -INFLUENCE. We start with the *basic reduction* and modify  $G'$  and the function  $\text{thr}$  as follows. Replace every edge  $\{v^t, v^b\}$  by an  $(k+2)$ -edge between  $v^t$  and  $v^b$ . Moreover, for a given constant  $\beta = \frac{8-5\varepsilon}{\varepsilon}$ , let  $L = \lceil n^\beta \rceil$  and we add  $nL$  more vertices  $x_1^1, \dots, x_n^1, \dots, x_1^L, \dots, x_n^L$ . For  $i = 1, \dots, n$ , vertex  $x_i^1$  is adjacent to all the bottom vertices. Moreover, for any  $j = 2, \dots, L$ , each  $x_i^j$  is adjacent to  $x_k^{j-1}$ , for any  $i, k \in \{1, \dots, n\}$ . We also add a vertex  $w$  and an  $n + (k+2)(\deg_G(v) - 1)$ -edge between  $w$  and  $v^b$ , for any bottom vertex  $v^b$ . For  $i = 1, \dots, n$ , vertex  $x_i^1$  is adjacent to  $w$ . For  $i = 1, \dots, n$  add  $n$  pending-vertices (*i.e.* degree one vertices) adjacent to  $x_i^L$ . For any vertex  $v^t$  add  $(\deg_G(v) + 1)(k+2)$  pending-vertices adjacent to  $v^t$ . Add also



**Fig. 1.** The graph  $G'$  obtained after carrying out the modifications of Theorem 1. A thick edge represents an  $\ell$ -edge for some  $\ell > 0$ . A “star” vertex  $v$  represents a vertex adjacent to  $\frac{\deg_{G'}(v)}{2}$  pending-vertices.

$n + n^2 + (k+2)(2m-n)$  pending-vertices adjacent to  $w$ . All vertices of the graph  $G'$  have the majority thresholds (see also Figure 1).

We claim that if  $I$  is a yes-instance then  $\text{opt}(I') \geq nL \geq n^{\beta+1}$ ; otherwise  $\text{opt}(I') < n^4$ . Let  $n' = |V'|$ , notice that we have  $n' \leq n^4 + nL$ .

Suppose that there exists a dominating set  $S \subseteq V$  in  $G$  of size at most  $k$ . Consider the solution  $S'$  for  $I'$  containing the corresponding top vertices and vertex  $w$ . After the first round, all vertices belonging to the edge gadgets which top vertex is in  $S'$  are activated. Since  $S$  is a dominating set in  $G$ , after the second round, all the bottom vertices are activated. Indeed  $\deg_{G'}(v^b) = 2(n + (k+2)\deg_G(v))$  and after the first round  $v^b$  has at least  $k+2$  neighbors activated belonging to an  $(k+2)$ -edge between  $v^b$  and some  $u^t \in V$  and  $n+(k+2)(\deg_G(v)-1)$  neighbors activated belonging to an  $n + (k + 2)(\deg_G(v) - 1)$ -edge between  $v^b$  and  $w$ . Thus, every vertex  $x_i^1$  gets active after the third round, and generally after the  $j$ th round,  $j = 4, \dots, L+2$  the vertices  $x_i^{j-2}$  are activated, and at the  $(L+3)$ th round all pending-vertices adjacent to  $x_i^L$  are activated. Therefore, the size of an optimal solution is at least  $nL \geq n^{\beta+1}$ .

Suppose that there is no dominating set in  $G$  of size  $k$ . Without loss of generality, we may assume that no pending-vertices are in a solution of  $I'$  since they all have threshold one. If  $w$  does not take part of a solution in  $I'$ , then no vertex  $x_i^1$  could be activated and in this case  $\text{opt}(I')$  is less than  $n' - nL \leq n^4$ . Consider now the solutions of  $I'$  of size  $k+1$  that contain  $w$ . Observe that if a top-vertex  $v^t$  gets active through bottom-vertices then  $v^t$  can not activate any other bottom-vertices. Indeed, as a contradiction, suppose that  $v^t$  is adjacent to a non-activated bottom-vertex. It follows that  $v^t$  could not have been activated because of its threshold and that no pending-vertices are part of the solution, a contradiction. Notice also that it is not possible to activate a bottom vertex by selecting some  $x_i^1$  vertices since of their threshold. Moreover, since there is no dominating set of size  $k$ , any subset of  $k$  top vertices cannot activate all bottom vertices, therefore no vertex  $x_i^k$ ,  $i = 1, \dots, n, k = 1, \dots, L$  can be activated.

Hence, less than  $n' - nL$  vertices can be activated in  $G'$  and the size of an optimal solution is at most  $n^4$ .

Assume now that there is an fpt-time  $n^{1-\varepsilon}$ -approximation algorithm  $A$  for MAX CLOSED  $(k+1)$ -INFLUENCE with majority threshold. Thus, if  $I$  is a *yes*-instance, the algorithm gives a solution of value  $A(I') \geq \frac{n^{\beta+1}}{(n')^{1-\varepsilon}} > \frac{n^{\beta+1}}{n^{(1-\varepsilon)(\beta+5)}} = n^4$  since  $n' \leq n^4 + nL < n^5L$ . If  $I$  is a *no*-instance, the solution value is  $A(I') < n^4$ . Hence, the approximation algorithm  $A$  can distinguish in fpt-time between *yes*-instances and *no*-instances for DOMINATING SET implying that FPT = W[2] since this last problem is W[2]-hard [10].  $\square$

**Theorem 2.** *For any  $\varepsilon \in (0, \frac{1}{2})$ , MAX CLOSED  $k$ -INFLUENCE and MAX OPEN  $k$ -INFLUENCE with thresholds at most two cannot be approximated within  $n^{\frac{1}{2}-\varepsilon}$  in fpt-time w.r.t. parameter  $k$  even on bipartite graphs, unless FPT = W[2].*

Using Lemma 2, Theorem 1, and Theorem 2 we can deduce the following corollary.

**Corollary 1.** *For any strictly increasing function  $r$ , MAX CLOSED  $k$ -INFLUENCE and MAX OPEN  $k$ -INFLUENCE with thresholds at most two or majority thresholds cannot be approximated within  $r(k)$  in fpt-time w.r.t. parameter  $k$  unless FPT = W[2].*

## 4 Unanimity Thresholds

For the unanimity thresholds case, we will give some results on general graphs before focusing on bounded degree graphs and regular graphs.

### 4.1 General Graphs

In this section, we first show that, in the unanimity case,  $(k, \ell)$ -INFLUENCE is W[1]-hard w.r.t. parameter  $k + \ell$  and MAX OPEN  $k$ -INFLUENCE is not approximable within  $n^{1-\varepsilon}$  for any  $\varepsilon \in (0, 1)$  in polynomial time, unless NP = ZPP. However, if we are allowed to use fpt-time then MAX OPEN  $k$ -INFLUENCE with unanimity is  $r(n)$ -approximable in fpt-time w.r.t. parameter  $k$  for any strictly increasing function  $r$ .

**Theorem 3.**  *$(k, \ell)$ -INFLUENCE with unanimity thresholds is W[1]-hard w.r.t. the combined parameter  $(k, \ell)$  even for bipartite graphs.*

**Theorem 4.** *For any  $\varepsilon \in (0, 1)$ , MAX OPEN  $k$ -INFLUENCE with unanimity thresholds cannot be approximated within  $n^{1-\varepsilon}$  in polynomial time, unless NP = ZPP.*

**Theorem 5.** *MAX OPEN  $k$ -INFLUENCE and MAX CLOSED  $k$ -INFLUENCE with unanimity thresholds are  $2^k$ -approximable in polynomial time.*

Using Lemma 2 and Theorem 5 we directly get the following.

**Corollary 2.** *For any strictly increasing function  $r$ , MAX OPEN  $k$ -INFLUENCE and MAX CLOSED  $k$ -INFLUENCE with unanimity thresholds are  $r(n)$ -approximable in fpt-time w.r.t. parameter  $k$ .*

For example, MAX OPEN  $k$ -INFLUENCE is  $\log(n)$ -approximable in time  $O^*(2^{k^2})$ .

*Finding dense subgraphs.* In the following we show that MAX OPEN  $k$ -INFLUENCE with unanimity thresholds is at least as difficult to approximate as the DENSEST  $k$ -SUBGRAPH problem, that consists of finding in a graph a subset of vertices of cardinality  $k$  that induces a maximum number of edges. In particular, any positive approximation result for MAX OPEN  $k$ -INFLUENCE with unanimity would directly transfers to DENSEST  $k$ -SUBGRAPH.

**Theorem 6.** *For any strictly increasing function  $r$ , if MAX OPEN  $k$ -INFLUENCE with unanimity thresholds is  $r(n)$ -approximable in fpt-time w.r.t. parameter  $k$  then DENSEST  $k$ -SUBGRAPH is  $r(n)$ -approximable in fpt-time w.r.t. parameter  $k$ .*

Using Theorem 6 and Corollary 2, we have the following corollary, independently established in [3].

**Corollary 3.** *For any strictly increasing function  $r$ , DENSEST  $k$ -SUBGRAPH is  $r(n)$ -approximable in fpt-time w.r.t. parameter  $k$ .*

## 4.2 Bounded Degree Graphs and Regular Graphs

We show in the following that MAX OPEN  $k$ -INFLUENCE and thus MAX CLOSED  $k$ -INFLUENCE are constant approximable in polynomial time on bounded degree graphs with unanimity thresholds. Moreover, MAX CLOSED  $k$ -INFLUENCE and then MAX OPEN  $k$ -INFLUENCE have no polynomial-time approximation scheme even on 3-regular graphs if  $P \neq NP$ . Moreover, we show that  $(k, \ell)$ -INFLUENCE is in FPT w.r.t. parameter  $k$ .

**Lemma 4.** *MAX OPEN  $k$ -INFLUENCE and MAX CLOSED  $k$ -INFLUENCE with unanimity thresholds on bounded degree graphs are constant approximable in polynomial time.*

**Theorem 7.** *MAX OPEN  $k$ -INFLUENCE and MAX CLOSED  $k$ -INFLUENCE with unanimity thresholds have no polynomial-time approximation scheme even on 3-regular graphs for  $k = \theta(n)$ , unless  $P = NP$ .*

In Theorem 3 we showed that  $(k, \ell)$ -INFLUENCE with unanimity thresholds is W[1]-hard w.r.t. parameters  $k$  and  $\ell$ . In the following we give several fixed-parameter tractability results for  $(k, \ell)$ -INFLUENCE w.r.t. parameter  $k$  on regular graphs and bounded degree graphs with unanimity thresholds. First we show that using results of Cai *et al.* [5] we can obtain fixed-parameter tractable algorithms. Then we establish an explicit and more efficient combinatorial algorithm. Using [5] we can show:

**Theorem 8.**  *$(k, \ell)$ -INFLUENCE with unanimity thresholds can be solved in  $2^{O(k\Delta^3)}n^2 \log n$  time where  $\Delta$  denotes the maximum degree and in  $2^{O(k^2 \log k)}n \log n$  time for regular graphs.*

While the previous results use general frameworks to solve the problem, we now give a direct combinatorial algorithm for  $(k, \ell)$ -INFLUENCE with unanimity thresholds on bounded degree graphs. For this algorithm we need the following definition and lemma.

**Definition 1.** *Let  $(\alpha, \beta)$  be a pair of positive integers,  $G = (V, E)$  an undirected graph with unanimity thresholds, and  $v \in V$  a vertex. We call  $v$  a realizing vertex for the pair  $(\alpha, \beta)$  if there exists a vertex subset  $V' \subseteq N^{2\alpha-1}[v]$  of size  $|V'| \leq \alpha$  such that  $|\sigma(V')| \geq \beta$  and  $\sigma[V']$  is connected. Furthermore, we call  $\sigma[V']$  a realization of the pair  $(\alpha, \beta)$ .*

We show first that in bounded degree graphs the problem of deciding whether a vertex is a realizing vertex for a pair of positive integers  $(\alpha, \beta)$  is fixed-parameter tractable w.r.t. parameter  $\alpha$ .

**Lemma 5.** *Checking whether a vertex  $v$  is a realizing vertex for a pair of positive integers  $(\alpha, \beta)$  can be done in  $\Delta^{O(\alpha^2)}$  time, where  $\Delta$  is the maximum degree.*

Consider in the following the CONNECTED  $(k, \ell)$ -INFLUENCE problem that is  $(k, \ell)$ -INFLUENCE with the additional requirement that  $G[\sigma[S]]$  has to be connected. Note that with Lemma 5 we can show that CONNECTED  $(k, \ell)$ -INFLUENCE is fixed parameter tractable w.r.t. parameter  $k$  on bounded degree graphs. Indeed, observe that two vertices in  $\sigma(S)$  cannot be adjacent since we consider unanimity thresholds. From this and the requirement that  $G[\sigma[S]]$  is connected, it follows that  $G[\sigma[S]]$  has a diameter of at most  $2k$ . Hence, the algorithm for CONNECTED  $(k, \ell)$ -INFLUENCE checks for each vertex  $v \in V$  whether  $v$  is a realizing vertex for the pair  $(k, \ell)$ . By Lemma 5 this gives an overall running time of  $\Delta^{O(k^2)} \cdot n$ .

We can extend the algorithm for the connected case to deal with the case where  $G[\sigma[S]]$  is not connected. The general idea is as follows. For each connected component  $C_i$  of  $G[\sigma[S]]$  the algorithm guesses the number of vertices in  $S \cap C_i$  and in  $\sigma(S) \cap C_i$ . This gives an integer pair  $(k_i, \ell_i)$  for each connected component in  $G[\sigma[S]]$ . Similar to the connected case, the algorithm will determine realizations for these pairs and the union of these realizations give  $S$  and  $\sigma(S)$ . Unlike the connected case, it is not enough to look for just one realization of a pair  $(k_i, \ell_i)$  since the realizations of different pairs may be not disjoint and, thus, vertices may be counted twice as being activated. To avoid the double-counting we show that if there are “many” different realizations for a pair  $(k_i, \ell_i)$ , then there always exist a realization being disjoint to all realizations of the other pairs. Now consider only the integer pairs that do not have “many” different realizations. Since there are only “few” different realizations possible, the graph induced by all the vertices contained in all these realizations is “small”. Thus, the algorithm can guess the realizations of the pairs having

---

**Algorithm 1.** The pseudocode of the algorithm solving the decision problem  $(k, \ell)$ -INFLUENCE. The guessing part in the algorithm behind Lemma 5 is used in Line 7 as subroutine. The final check in Line 19 is done by brute force checking all possibilities.

---

```

1: procedure SOLVEINFLUENCE( $G$ , thr,  $k$ ,  $\ell$ )
2:   Guess  $x \in \{1, \dots, k\}$             $\triangleright x$ : number of connected components of  $G[\sigma[S]]$ 
3:   Guess  $(k_1, \ell_1), \dots, (k_x, \ell_x)$  such that  $\sum_{i=1}^x k_i = k$  and  $\sum_{i=1}^x \ell_i = \ell$ 
4:   Initialize  $c_1 = c_2 = \dots = c_x \leftarrow 0$      $\triangleright$  one counter for each integer pair  $(k_i, \ell_i)$ 
5:   for each vertex  $v \in V$  do           $\triangleright$  determine realizing vertices
6:     for  $i \leftarrow 1$  to  $x$  do
7:       if  $v$  is a realizing vertex for the pair  $(k_i, \ell_i)$  then       $\triangleright$  see Lemma 5
8:          $c_i \leftarrow c_i + 1$ 
9:          $T(v, i) = \text{"yes"}$ 
10:      else
11:         $T(v, i) = \text{"no"}$ 
12:      initialize  $X \leftarrow \emptyset$            $\triangleright X$  stores all pairs with "few" realizations
13:      for  $i \leftarrow 1$  to  $x$  do
14:        if  $c_i \leq 2 \cdot x \cdot \Delta^{4k}$  then
15:           $X \leftarrow X \cup \{i\}$ 
16:      for each vertex  $v \in V$  do       $\triangleright$  remove vertices not realizing any pair in  $X$ 
17:        if  $\forall i \in X : T(v, i) = \text{"no"}$  then
18:          delete  $v$  from  $G$ .
19:      if all pairs  $(k_i, \ell_i)$ ,  $i \in X$ , can be realized in the remaining graph then
20:        return 'YES'
21:      else
22:        return 'NO'
```

---

only "few" realizations and afterwards add greedily disjoint realizations of pairs having "many" realizations. See Algorithm 1 for the pseudocode.

**Theorem 9.** *Algorithm 1 solves  $(k, \ell)$ -INFLUENCE with unanimity thresholds in  $2^{O(k^2 \log(k\Delta))} \cdot n$  time, where  $\Delta$  is the maximum degree of the input graph.*

## 5 Conclusions

We established results concerning the parameterized complexity as well as the polynomial-time and fpt-time approximability of two problems modeling the spread of influence in social networks, namely MAX OPEN  $k$ -INFLUENCE and MAX CLOSED  $k$ -INFLUENCE.

In the case of unanimity thresholds, we show that MAX OPEN  $k$ -INFLUENCE is at least as hard to approximate as DENSEST  $k$ -SUBGRAPH, a well-studied problem. We established that DENSEST  $k$ -SUBGRAPH is  $r(n)$ -approximable for any strictly increasing function  $r$  in fpt-time w.r.t. parameter  $k$ . An interesting open question consists of determining whether MAX OPEN  $k$ -INFLUENCE is constant approximable in fpt-time. Such a positive result would improve the approximation in fpt-time for DENSEST  $k$ -SUBGRAPH. In the case of thresholds

bounded by two we excluded a polynomial time approximation scheme for MAX CLOSED  $k$ -INFLUENCE but we did not find any polynomial-time approximation algorithm. Hence, the question arises, whether this hardness result can be strengthened. Another interesting open question is to study the approximation of min target set selection problem in fpt-time.

## References

1. Aazami, A., Stilp, K.: Approximation algorithms and hardness for domination with propagation. *SIAM J. Discrete Math.* 23(3), 1382–1399 (2009)
2. Ben-Zwi, O., Hermelin, D., Lokshtanov, D., Newman, I.: Treewidth governs the complexity of target set selection. *Discrete Optim.* 8(1), 87–96 (2011)
3. Bourgeois, N., Giannakos, A., Lucarelli, G., Milis, I., Paschos, V.T.: Exact and approximation algorithms for densest  $k$ -subgraph. In: Ghosh, S.K., Tokuyama, T. (eds.) WALCOM 2013. LNCS, vol. 7748, pp. 114–125. Springer, Heidelberg (2013)
4. Cai, L.: Parameterized complexity of cardinality constrained optimization problems. *Comput. J.* 51(1), 102–121 (2008)
5. Cai, L., Chan, S.M., Chan, S.O.: Random separation: A new method for solving fixed-cardinality optimization problems. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 239–250. Springer, Heidelberg (2006)
6. Chang, C.-L., Lyuu, Y.-D.: Spreading messages. *Theor. Comput. Sci.* 410(27–29), 2714–2724 (2009)
7. Chen, N.: On the approximability of influence in social networks. *SIAM J. Discrete Math.* 23(3), 1400–1415 (2009)
8. Chopin, M., Nichterlein, A., Niedermeier, R., Weller, M.: Constant thresholds can make target set selection tractable. In: Even, G., Rawitz, D. (eds.) MedAlg 2012. LNCS, vol. 7659, pp. 120–133. Springer, Heidelberg (2012)
9. Dinur, I., Safra, S.: The importance of being biased. In: Proc. of STOC, pp. 33–42. ACM (2002)
10. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer (1999)
11. Dreyer, P.A., Roberts, F.S.: Irreversible  $k$ -threshold processes: Graph-theoretical threshold models of the spread of disease and of opinion. *Discrete Appl. Math.* 157(7), 1615–1627 (2009)
12. Kempe, D., Kleinberg, J., Tardos, É.: Maximizing the spread of influence through a social network. In: Proc. of KDD, pp. 137–146. ACM (2003)
13. Marx, D.: Parameterized complexity and approximation algorithms. *Comput. J.* 51(1), 60–78 (2008)
14. Nichterlein, A., Niedermeier, R., Uhlmann, J., Weller, M.: On tractable cases of target set selection. *Soc. Network Anal. Mining* (2012) (online available)
15. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press (2006)
16. Peleg, D.: Local majorities, coalitions and monopolies in graphs: a review. *Theor. Comput. Sci.* 282, 231–257 (2002)
17. Reddy, T.V.T., Rangan, C.P.: Variants of spreading messages. *J. Graph Algorithms Appl.* 15(5), 683–699 (2011)

# An Effective Branching Strategy for Some Parameterized Edge Modification Problems with Multiple Forbidden Induced Subgraphs\*

Yunlong Liu<sup>1,2</sup>, Jianxin Wang<sup>1</sup>, Chao Xu<sup>1</sup>, Jiong Guo<sup>3</sup>, and Jianer Chen<sup>1,4</sup>

<sup>1</sup> School of Information Science and Engineering, Central South University,  
Changsha 410083, P.R. China

<sup>2</sup> College of Mathematics and Computer Science, Key Laboratory of High Performance Computing and Stochastic Information Processing(Ministry of Education of China), Hunan Normal University,  
Changsha 410081, P.R. China

<sup>3</sup> Universität des Saarlandes, Campus E 1.7,  
D-66123 Saarbrücken, Germany

<sup>4</sup> Department of Computer Science and Engineering, Texas A&M University,  
College Station, TX 77843, USA  
 [{hnsdlyl,xuchaofay}@163.com](mailto:{hnsdlyl,xuchaofay}@163.com), [jxwang@mail.csu.edu.cn](mailto:jxwang@mail.csu.edu.cn),  
[jguo@mmci.uni-saarland.de](mailto:jguo@mmci.uni-saarland.de), [chen@cse.tamu.edu](mailto:chen@cse.tamu.edu)

**Abstract.** Branching on forbidden induced subgraphs is a genetic strategy to obtain parameterized algorithms for many edge modification problems. For such a problem in which the graph property is defined by multiple forbidden induced subgraphs, branching process is trivially performed on each subgraph. Thus, the size of the resulting search tree is dominated by the size of the largest forbidden subgraph. In this paper, we present a simple strategy for deriving significantly improved branching rules for dealing with multiple forbidden subgraphs by edge modifications. The basic idea hereby is that while constructing branching rules for the largest forbidden subgraph, we sufficiently take into account the structural relationship between it and other forbidden subgraphs. By applying this strategy, we obtain improved parameterized algorithms for edge modification problems for several graph properties such as proper interval, 3-leaf power, threshold and co-trivially perfect graphs.

## 1 Introduction

Edge modification problems call for making a minimum number of changes to the edge set of an input graph in order to obtain a graph with a desired property. These problems play an important role in computer science and have applications in several fields, such as molecular biology, numerical algebra and search

---

\* This research was supported in part by the National Natural Science Foundation of China under Grant No.61070224, No.61232001, and No.61128006, the China Postdoctoral Science Foundation funded project under Grant No. 2012M521551, and the DFG Cluster of Excellence “Multimodal Computing and Interaction (MMCI)”.

games [13]. For NP-hard edge modification problems, studying their parameterized complexity has received much attention in recent years. Positively, many edge modification problems have been shown to be *fixed-parameter tractable* (abbreviated by FPT).

Until now, most parameterized algorithms for edge modification problems with graph properties, which can be characterized by forbidden induced subgraphs, are based on a bounded search tree method, and the size of the search tree usually dominates the computation time. Decreasing the size of the search tree has all the way been a focus in the field of parameterized computation and has received considerable attention. The basic approach stems from extensive case distinction based on the neighboring structures of the forbidden subgraphs. The most notable effort along this line of research is the automated search tree generation introduced by Gramm et al. [6]. Its main steps include enumerating all “relevant” subgraphs containing the forbidden subgraphs and checking all possible branching rules for every enumerated subgraph. The implementation of this complicated process relies on the computational power of computers in order to generate and analyze the search tree. Another approach for improving branching on the forbidden subgraphs is to study relaxations of the target graph classes [7,10,11,14]. Hereby, one has to search for a relaxed graph class such that optimally modifying a graph from this class to the target graph class can be done in polynomial time.

For an edge modification problem for which the graph property is defined by multiple forbidden subgraphs, the common approach is to apply the trivial branching to each of the forbidden subgraphs separately. Thus, the size of the resulting search tree is dominated by the largest forbidden subgraph. Therefore, more effective branching rules for the largest forbidden subgraph can significantly decrease the search tree size and then the overall running time of the algorithm. However, a more refined case distinction based on neighboring structures of the forbidden subgraphs could dramatically increase the number of cases to consider. The analysis of the branching rules becomes more and more complex. Therefore, in most works dealing with multiple forbidden subgraphs, only trivial branching rules usually are performed to avoid the tedious analysis.

Given a graph property with multiple forbidden subgraphs, we can often observe that there are some structural connections between the forbidden subgraphs. More precisely, after deleting or inserting an edge in the largest forbidden subgraph, another forbidden subgraph may be induced. To destroy the new forbidden subgraph, another round of branching has to be performed. Especially, for some problems with at least three forbidden induced subgraphs, some of forbidden subgraphs may be chain-induced and a chain of branchings has to be performed. Combining such chain-branchings could lead to a significantly improved branching rule for the largest forbidden subgraph. Based on this observation, we develop a general strategy that sufficiently considers the structural relationship among distinct forbidden subgraphs. To demonstrate the power of this strategy, we present new search tree algorithms for several edge modification problems and obtain improved search tree sizes correspondingly (see Table 1).

**Table 1.** Comparison of the current results and ours

| Problems                           | The Current Results | Ref. | Our Results          |
|------------------------------------|---------------------|------|----------------------|
| proper interval edge deletion      | $O(9^k n^{O(1)})$   | [14] | $O(6^k n^{O(1)})$    |
| 3-leaf power edge deletion         | $O(6^k nm)$         | [5]  | $O(5^k + n + m)$     |
| threshold edge deletion            | $O(4^k + kn^4)$     | [8]  | $O(2^k + kn^4)$      |
| co-trivially perfect edge deletion | $O(3^k + kn^4)$     | [8]  | $O(2^k + kn^4)$      |
| proper interval edge insertion     | $O(16^k m)$         | [9]  | $O(4^k + nm(n + m))$ |
| 3-leaf power edge insertion        | $O((k + 3)^k nm)$   | [5]  | $O(4^k + n + m)$     |

This general strategy seems promising for obtaining efficient branching rules for various other problems with multiple forbidden induced subgraphs.

We mention that in the prior study of the trivially-perfect edge deletion problem by the branching strategy based on graph class relaxation, in a specific procedure of getting relaxed  $P_4$ -sparse graphs, Nastos et al. [11] also observed that deleting any edge from a forbidden subgraph  $C_4$  exactly results in another forbidden subgraph  $P_4$ , and thus branched on 6 ways to destroy  $C_4$ .

## 2 Terminology and Notations

We consider only simple and undirected graphs. For a graph  $G = (V, E)$ , let  $n = |V|$  and  $m = |E|$ . For two vertices  $x$  and  $y$ , let  $(x, y)$  denote the edge between  $x$  and  $y$ . Inserting an edge  $e$  to  $G$  and deleting an edge  $e$  from  $G$  are denoted by  $G + e$  and  $G - e$ , respectively. We use  $+e$  to denote the insertion of edge  $e$  and use  $-e$  to denote the deletion of  $e$ . A subgraph of  $G$  induced by a set  $V' \subset V$  is denoted by  $G[V'] = (V', E')$ , where  $E' = \{(u, v) \mid (u, v) \in E \wedge u, v \in V'\}$ .

A graph  $G$  is  $F$ -free for the graph  $F$  if  $G$  does not contain an induced subgraph isomorphic to  $F$ . If a graph class  $\mathcal{G}$  is  $\mathcal{F}$ -free for a set  $\mathcal{F}$  of some graphs, then the graphs in  $\mathcal{F}$  are called the forbidden induced subgraphs of  $\mathcal{G}$ . Moreover, if  $|\mathcal{F}| > 1$ , then  $\mathcal{G}$  is called a graph class with multiple forbidden induced subgraphs.

## 3 The General Technique

For the edge modification problems with multiple forbidden induced subgraphs, search tree algorithms based on trivial branching basically consist of a set of branching rules, each of which corresponds to one subgraph and consists of all possible cases to destroy the subgraph. To obtain more efficient branching rules, we mainly take advantage of the underlying relationship among distinct subgraphs. The idea behind our strategy is roughly described as follows.

First, specify the finite set  $\mathcal{F}$  of considered forbidden induced subgraphs. For infinite forbidden induced subgraphs, sometimes only finite of them are necessarily considered. Our aim is to design refined branching rules for some subgraphs in  $\mathcal{F}$  for which there are most modification possibilities to consider. Then, for a determined subgraph  $U \in \mathcal{F}$ , produce a refined branching rule. This procedure

is accomplished by several possible rounds of branchings. After each round of branching, check whether there exists other induced subgraph  $f \in \mathcal{F}$  in the resulting graphs. If so, then specify  $f$  as the branching object of the next round. Otherwise, the branching process for this subgraph is finished. During each round of branching, we apply the trivial branching to the branching object.

In the following, we describe the implementation of our strategy.

Given a set  $\mathcal{F}$  of finite forbidden induced subgraphs and an arbitrary subgraph  $U \in \mathcal{F}$ , the procedure of constructing an refined branching rule on  $U$  is described in Fig. 1. Assume that the edge modification operation is edge deletion. For the case of edge insertion, the depiction is similar.

To avoid generating redundant subcases during the branching, we annotate some edges in the resulting graphs with the label “forbidden”. A forbidden edge means that it has been considered in one case and can be omitted in other cases. Note that after performing the *CombineBranch*-procedure, the repeated cases should be deleted and only the minimal cases are kept in the resulted branching rule.

**Theorem 1.** *Given an edge modification problem  $\mathcal{P}$  with a set  $\mathcal{F}$  of forbidden induced subgraphs, and an arbitrary subgraph  $U \in \mathcal{F}$ , then the procedure *CombineBranch* produces a sound branching rule for the subgraph  $U$ .*

*Proof.* Assume that  $I$  is an instance of problem  $\mathcal{P}$ ,  $S$  is a solution for  $I$ , and  $i$  is the number of possible branching rounds in *CombineBranch*. We prove this theorem by induction on  $i$ . The initial case  $i = 1$  is obvious: a subgraph  $U$  can always be branched on itself trivially.

Consider a general number  $i \geq 1$ . Suppose that after the  $i$ -th round of branching, the branching cases on  $U$  are  $B_1, B_2, \dots, B_r$ , and the corresponding resulting subgraphs are  $C_1, C_2, \dots, C_r$ . This means that one of cases  $B_1, B_2, \dots, B_r$  must be in  $S$ . W.l.o.g., suppose that in the resulting graph  $C_j$  ( $1 \leq j \leq r$ ), there exists at least one forbidden induced subgraph  $f \in \mathcal{F}$ . To destroy the subgraph  $f$ , another round of branching has to be performed. In the following, we analyze the  $(i + 1)$ -th round of branching.

Let the trivial branching cases for destroying  $f$  be  $\{-e_1\}, \{-e_2\}, \dots, \{-e_m\}$  (excluding the forbidden cases). In the procedure *CombineBranch*,  $f$  is exactly branched into these trivial cases. It is obvious that one of cases  $B_1, B_2, \dots, B_j \cup \{-e_1\}, B_j \cup \{-e_2\}, \dots, B_j \cup \{-e_m\}, B_{j+1}, \dots, B_r$  must be in  $S$ . Therefore, after the  $(i + 1)$ -th round of branching, the cases produced by the procedure *CombineBranch* constitute a refined branching rule for  $U$ .

Next, we argue that the labeled cases in one branching object can be omitted safely. W.l.o.g, assume that during the  $(i + 1)$ -th round of branching,  $f$  is the branching object, and  $\{-e_j\}$  is a trivial case in which the edge  $e_j$  labeled “forbidden”. According to the procedure *CombineBranch*, the edge  $e_j$  was labeled “forbidden” means that during the  $i$ -th round of branching,  $e_j$  has already been considered in one minimal branching case. Hence, during the  $(i + 1)$ -th round of branching,  $\{-e_j\}$  can be omitted safely.  $\square$

**Procedure CombineBranch( $U, \mathcal{F}$ )**

Input: A subgraph  $U$  and a collection  $\mathcal{F}$  of forbidden structures ;

Output: A branching rule on  $U$  ; /\* a branching rule is denoted by a family of edge modification sets \*/

**Method:**

```

if  $U$  does not contain any induced subgraph in  $\mathcal{F}$  then
    Return  $\mathcal{Q} = \{\emptyset\}$ ;
else
    Pick a subgraph  $f \in \mathcal{F}$  that is contained in  $U$ ;
    Apply the trivial branching to  $f$  excluding the forbidden edges;
    Let the deleted edges be  $e_1, \dots, e_m$  ;
    Let the corresponding resulting graphs be  $C_1, \dots, C_m$  ;
    Set  $D := \emptyset$  ;
    for  $i = 1$  to  $m$  do
        if  $C_i$  does not contain any induced subgraph in  $\mathcal{F}$  then
            { Set flag  $w_i := 0$  ;
              Add the corresponding deleted edge  $e_i$  to  $D$  ; }
        else
            Set flag  $w_i := 1$  ;
        Set  $\mathcal{Q} := \emptyset$  ;
        for  $i = 1$  to  $m$  do
            { if  $w_i = 1$  then
                label the edges “forbidden” on  $C_i$  according to the edge set  $D$  ;
                CombineBranch( $C_i, \mathcal{F}$ ) ;
                Let the returned collection be  $\mathcal{Q}_i$  ;
                if  $\mathcal{Q}_i \neq \emptyset$  then
                    for each set  $B$  in  $\mathcal{Q}_i$  do add  $B \cup \{-e_i\}$  to  $\mathcal{Q}$  ; }
        Return  $\mathcal{Q}$ .
    
```

**Fig. 1.** The procedure for constructing branching rule

Observe that the number of the trivial cases for the “smallest” subgraph is the threshold value of the search tree algorithm. If a subgraph already yields a branching number better than the threshold value, we proceed with the next subgraph. This process can be implemented by setting proper forbidden labels. For some problems with at least three forbidden induced subgraphs, considering threshold value may result in concise branching rule.

Compared to the case distinction based on neighboring structures of forbidden subgraphs mentioned above, our strategy has two advantages. First, we need not to introduce any extended subgraph, and any substructure to be considered is not more complicated than the “largest” forbidden subgraph. Second, the improvement on the branching number is obvious.

Our strategy can be applied not only to the problems with finite forbidden induced subgraphs, but also to some problems with infinite forbidden induced subgraphs, from which only finite forbidden induced subgraphs need to be considered.

## 4 Applications and Results

We apply the general strategy to several edge modification problems.

### 4.1 Edge Deletion Problems

We discuss four edge deletion problems: proper interval edge deletion, 3-leaf power edge deletion, threshold edge deletion, and co-trivially perfect edge deletion.

**Proper Interval Edge Deletion.** A graph is a *proper interval graph* if and only if it is  $\{\text{claw}, \text{net}, \text{tent}, \text{hole}\}$ -free [15]. *Claw*, *net* and *tent* are graphs containing at most 6 vertices depicted in Fig. 2, and *hole* is an induced cycle of length at least 4. Furthermore, we use  $C_r$  to denote a hole of length  $r \geq 4$ .



**Fig. 2.** The (infinite) family of forbidden induced subgraphs of proper interval graphs

Proper Interval Edge Deletion (PIED) is defined as follows:

**Input:** An undirected graph  $G = (V, E)$  and a positive integer  $k$ .

**Parameter:**  $k$

**Task:** Find a set  $F \subseteq E$  of size at most  $k$  such that  $H = (V, E \setminus F)$  is a proper interval graph or answer “No”.

For the PIED problem, there exists a polynomial-time algorithm to solve it on  $\{\text{claw}, \text{net}, \text{tent}, C_4, C_5, C_6\}$ -free graphs. Combining this with branchings on the forbidden structures *claw*, *net*, *tent*,  $C_4$ ,  $C_5$ , and  $C_6$  results in a parameterized algorithm with running time  $O(9^k n^{O(1)})$  [14].

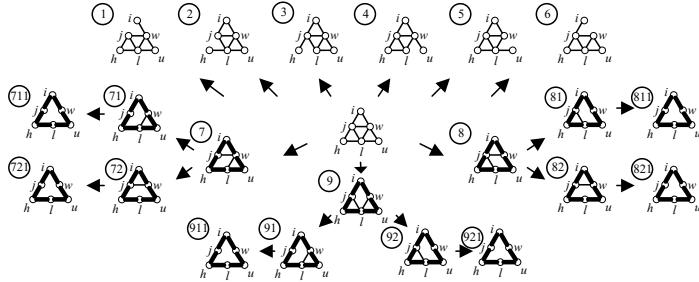
We improve this search tree algorithm with our strategy. Since the forbidden subgraph *tent* with the most edge deletion possibilities, we focus on dealing with it. Our task is to derive a refined branching rule on *tent* by exploiting the relationship between it and  $C_4$ ,  $C_5$ ,  $C_6$ .

**Branching Rule 1.** Given a *tent*  $T = (V, E)$ , in which  $V = \{i, j, h, l, u, w\}$  (see Fig.3). For the PIED problem, one can branch it into six cases:  $\{-(i, j)\}$ ,  $\{-(j, h)\}$ ,  $\{-(h, l)\}$ ,  $\{-(l, u)\}$ ,  $\{-(u, w)\}$ , and  $\{-(w, i)\}$ .

**Lemma 1.** *Branching Rule 1 is safe, and the branching number is 6.*

**Theorem 2.** *The PIED problem can be solved in  $O(6^k n^{O(1)})$  time.*

Note that if we apply our strategy as the procedure in Fig. 1 to all other forbidden subgraphs, we can obtain a branching rule with a branching number at most 3.11. However, for the purpose of demonstrating the applicability and flexibility of our strategy, it suffices to show the improvement of the branching number from 9 to 6.



**Fig. 3.** A branching way for Tent (Bold lines denote forbidden edges)

**3-Leaf Power Edge Deletion.** For an unrooted tree  $T$  with leaves one-to-one labeled by the elements of a set  $V$ , the 3-leaf power of  $T$  is a graph, denoted by  $T^3$ , with  $T^3 = (V, E)$ , where  $E = \{(u, v) | u, v \in V \text{ and } d(u, v) \leq 3\}$ , in which  $d(u, v)$  denotes the length of the path between  $u$  and  $v$  in  $T$ . See Fig.4 for a depiction of the forbidden induced subgraphs of 3-leaf power graphs [5].



**Fig. 4.** The (infinite) family of forbidden induced subgraphs of 3-leaf power graphs

3-Leaf Power Edge Deletion (3LPED) is defined as follows:

**Input:** An undirected graph  $G = (V, E)$  and a positive integer  $k$ .

**Parameter:**  $k$

**Task:** Find a set  $F \subseteq E$  of size at most  $k$  such that  $H = (V, E \setminus F)$  is a  $T^3$ -graph for a tree  $T$  or answer “No”.

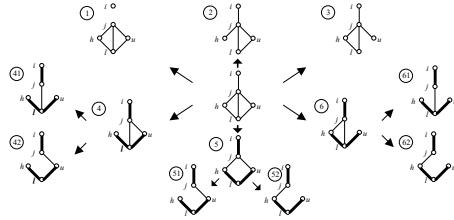
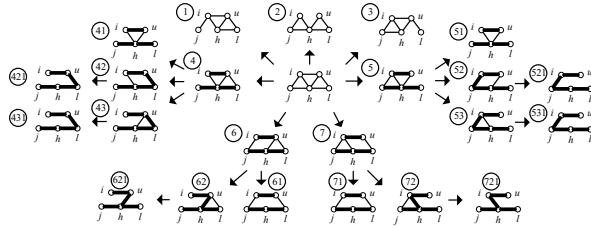
For the 3LPED problem, there exists a polynomial-time algorithm to solve it on  $\{\text{bull}, \text{dart}, \text{gem}, C_4\}$ -free graphs. Based on this observation, Dom et al. presented an FPT algorithm with running time  $O(6^k nm)$  [5].

We also improve this algorithm with our strategy. Branch on the forbidden subgraphs *bull*, *dart*, *gem*,  $C_4$ , and then combine the polynomial procedure. It remains to derive refined branching rules for the forbidden structures *dart* and *gem*, respectively.

**Branching Rule 2.** Let  $D = (V, E)$  be a *dart* with  $V = \{i, j, h, l, u\}$  (see Fig.5). For the 3LPED problem, one can branch it into 6 cases:  $\{-(i, j)\}$ ,  $\{-(h, l)\}$ ,  $\{-(l, u)\}$ ,  $\{-(j, h), -(j, u)\}$ ,  $\{-(j, h), -(j, l)\}$ , and  $\{-(j, l), -(j, u)\}$ .

**Lemma 2.** *Branching Rule 2 is safe, and the branching number is 3.8 or better.*

**Branching Rule 3.** Let  $G = (V, E)$  be a *Gem* with  $V = \{i, j, h, l, u\}$  (see Fig.6). For the 3LPED problem, one can branch it into 6 cases:  $\{-(j, h)\}$ ,  $\{-(i, u)\}$ ,  $\{-(h, l)\}$ ,  $\{-(i, h), -(h, u)\}$ ,  $\{-(i, h), -(i, j), -(u, l)\}$ , and  $\{-(i, j), -(u, l), -(h, u)\}$ .

**Fig. 5.** The branching procedure on Dart**Fig. 6.** The branching procedure on Gem

**Lemma 3.** *Branching Rule 3 is safe, and the branching number is 3.5 or better.*

**Lemma 4.** [1] *3-Leaf Power Edge Deletion admits a polynomial kernel, and the kernelization procedure can be done in linear time.*

Employing the interleaving techniques in [12], we can obtain the following theorem.

**Theorem 3.** *3-Leaf Power Edge Deletion can be solved in  $O(5^k + n + m)$  time.*

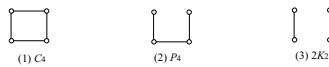
**Threshold Edge Deletion and Co-trivially Perfect Edge Deletion.** A graph  $G$  is a threshold graph iff  $G$  contains no induced  $2K_2$ ,  $C_4$ , and  $P_4$ , while a co-trivially perfect graph contains no induced  $2K_2$  and  $P_4$  [3]. See Fig.7 for a depiction of the forbidden induced subgraphs of threshold graphs.

Threshold Edge Deletion(TED) is defined as follows:

**Input:** An undirected graph  $G = (V, E)$  and a positive integer  $k$ .

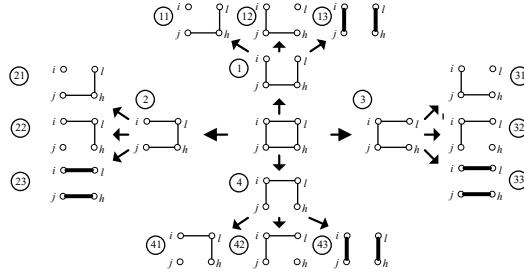
**Parameter:**  $k$

**Task:** Find a set  $F \subseteq E$  of size at most  $k$  such that  $H = (V, E \setminus F)$  is a threshold graph or answer “No”.

**Fig. 7.** The forbidden induced subgraphs of threshold graphs

For the TED problem, there exists an FPT algorithm with running time  $O(4^k + kn^4)$  in [8]. We improve this algorithm by deriving refined branching rules for  $C_4$  and  $P_4$  respectively.

**Branching Rule 4.** Let  $(V, E)$  be a  $C_4$  with  $V = \{i, j, h, l\}$  (see Fig.8). For the TED problem, one can branch it into 4 cases:  $\{-(i, j), -(i, l)\}$ ,  $\{-(i, l), -(l, h)\}$ ,  $\{-(l, h), -(j, h)\}$ , and  $\{-(i, j), -(j, h)\}$ .



**Fig. 8.** The branching procedure on  $C_4$  (implicitly including  $P_4$  )

**Lemma 5.** *Branching Rule 4 is safe, and the branching number is 2.*

**Branching Rule 5.** Let  $(V, E)$  be a  $P_4$  with  $V = \{i, j, h, l\}$  (see Fig.8 ①). For the TED problem, one can branch it into 2 cases:  $\{-(i, j)\}$ , and  $\{-(h, l)\}$ .

**Lemma 6.** *Branching Rule 5 is safe, and the branching number is 2.*

**Theorem 4.** *Threshold Edge Deletion can be solved in  $O(2^k + kn^4)$  time.*

**Theorem 5.** *Co-Trivially Perfect Edge Deletion can be solved in  $O(2^k + kn^4)$  time.*

## 4.2 Edge Insertion Problems

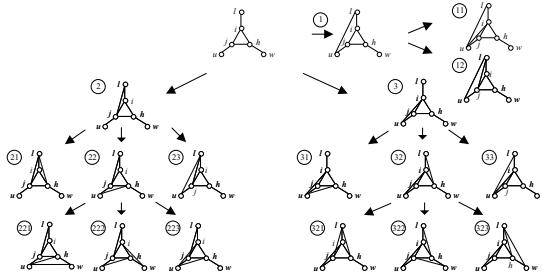
**Proper Interval Edge Insertion.** The Proper Interval Edge Insertion (PIEI) problem is defined as follows:

**Input:** An undirected graph  $G = (V, E)$  and a positive integer  $k$ .

**Parameter:**  $k$

**Task:** Find a set  $F$  of at most  $k$  edges such that  $G + F = (V, E \cup F)$  is a proper interval graph or answer “No”.

The study of parameterized algorithms for PIEI was initiated by Kaplan et al. [9]. Furthermore, Kaplan et al. presented a search tree algorithm of running time  $O^*(16^k)$ . The main part of this algorithm is dealing with the holes by triangulation. In fact, destroying holes by triangulation can be done in  $O^*(4^k)$  time [4]. We focus here on deriving refined branching rules for Net and Tent.



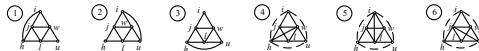
**Fig. 9.** Refined branching on the forbidden structure Net (partly)

Destroying one Net  $N$  contains 9 trivial cases. However, for PIEI, after inserting one edge in  $N$ , other forbidden structures such as  $C_4$  or *claw* are chain-induced. To destroy these forbidden structures, a chain of branchings has to be performed. Combining these branchings will result in a refined branching rule.

**Branching Rule 6.** Let  $N$  be a Net induced by the vertices  $i, j, h, l, u, w$  (see Fig.9). For the PIEI problem, one can branch it into  $\{+(u, l), +(u, i)\}$ ,  $\{+(u, l), +(l, j)\}$ ,  $\{+(u, w), +(u, h)\}$ ,  $\{+(u, w), +(w, j)\}$ ,  $\{+(w, l), +(w, i)\}$ ,  $\{+(w, l), +(l, h)\}$ ,  $\{+(l, j), +(l, h)\}$ ,  $\{+(u, i), +(u, h)\}$ ,  $\{+(w, j), +(w, i)\}$ ,  $\{+(u, i), +(w, j), +(l, h)\}$ ,  $\{+(l, j), +(u, h), +(w, i)\}$ .

**Lemma 7.** *Branching Rule 6 is safe, and the branching number is 3.11 or better.*

To destroy a Tent  $T$ , there are six trivial cases of edge insertions (see Fig. 10). However, for the PIEI problem, we also get an important observation that three cases suffice and other cases can be omitted safely.



**Fig. 10.** Six trivial cases of edge insertion for Tent (Dashed lines denote forbidden edges)

**Branching rule 7.** Let  $(V, E)$  be a Tent with  $V = \{i, j, h, l, u, w\}$  (see Fig.10). For the PIEI problem, one can branch it into 3 cases:  $\{+(i, h)\}$ ,  $\{+(i, u)\}$ , and  $\{+(h, u)\}$ .

**Lemma 8.** *Branching Rule 7 is safe, and the branch number is 3.*

**Lemma 9.** [2] *Proper Interval Edge Insertion admits a polynomial kernel, and the kernelization procedure can be done in  $O(nm(n + m))$  time.*

**Theorem 6.** *Proper Interval Edge Insertion can be solved in  $O(4^k + nm(m + n))$  time.*

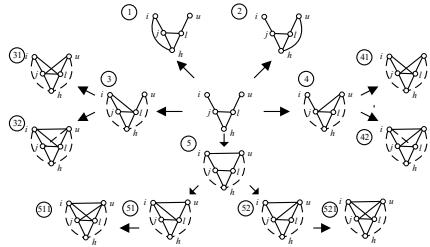
**3-Leaf Power Edge Insertion.** 3-Leaf Power Edge Insertion (3LPEI) is defined as follows:

**Input:** An undirected graph  $G = (V, E)$  and a positive integer  $k$ .

**Parameter:**  $k$

**Task:** Find a set  $F$  of at most  $k$  edges such that  $G + F = (V, E \cup F)$  is a  $T^3$ -graph for a tree  $T$  or answer “No”.

For the 3LPEI problem, Dom. et al. presented a parameterized algorithm with running time  $O((k+3)^k nm)$  [5]. Here, we present a search tree algorithm. For holes, we can deal with it by triangulation in  $O(4^k(n+m))$  time [4]. For a *dart*, it is trivially branched into 4 cases. And for a *gem*, it is trivially branched into 3 cases. We mainly focus on *bull*. Although there are 5 trivial cases, we can derive a refined branching rule for it.



**Fig. 11.** The branching procedure on Bull (Dashed lines denote forbidden edges)

**Branching Rule 8.** Let  $B = (V, E)$  be a *Bull* with  $V = \{i, j, h, l, u\}$  (see Fig.11). For the 3LPEI problem, one can branch it into 3 cases:  $\{+(i, h)\}$ ,  $\{+(u, h)\}$ , and  $\{+(i, l), +(u, j)\}$ .

**Lemma 10.** *Branching Rule 8 is safe, and the branch number is 2.5 or better.*

**Lemma 11.** [1] *3-Leaf Power Edge Insertion admits a polynomial kernel, and the kernelization procedure can be done in linear time.*

**Theorem 7.** *3-Leaf Power Edge Insertion can be solved in  $O(4^k + n + m)$  time.*

## 5 Conclusions

In this paper, we propose a simple strategy to generate refined branching rules for the edge modification problems, for which the graph properties are defined by multiple forbidden induced subgraph. Moreover, applying this strategy, we further improve the search tree sizes for several edge modification problems. We believe that our strategy can apply to many other edge modification problems.

## References

1. Bessy, S., Paul, C., Perez, A.: Polynomial Kernels for 3-Leaf Power Graph Modification Problems. *Discrete Applied Mathematics* 158(16), 1732–1744 (2010)
2. Bessy, S., Perez, A.: Polynomial kernels for proper interval completion and related problems. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 229–239. Springer, Heidelberg (2011)
3. Brandstädt, A., Le, V.B., Spinrad, J.P.: *Graph Classes: a Survey*. SIAM Monographs on Discrete Mathematics and Applications (1999)
4. Cai, L.: Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters* 58(4), 171–196 (1996)
5. Dom, M., Guo, J., Hüffner, F., Niedermeier, R.: Error compensation in leaf power problems. *Algorithmica* 44(4), 363–381 (2006)
6. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica* 39(4), 321–347 (2004)
7. Guo, J., Hüffner, F., Komusiewicz, C., Zhang, Y.: Improved algorithms for bicluster editing. In: Agrawal, M., Du, D.-Z., Duan, Z., Li, A. (eds.) TAMC 2008. LNCS, vol. 4978, pp. 445–456. Springer, Heidelberg (2008)
8. Guo, J.: Problem kernels for NP-complete edge deletion problems: Split and related graphs. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 915–926. Springer, Heidelberg (2007)
9. Kaplan, H., Shamir, R., Tarjan, R.E.: Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM J. Comput.* 28(5), 1906–1922 (1999)
10. Liu, Y., Wang, J., Guo, J., Chen, J.: Cograph editing: Complexity and parameterized algorithms. In: Fu, B., Du, D.-Z. (eds.) COCOON 2011. LNCS, vol. 6842, pp. 110–121. Springer, Heidelberg (2011)
11. Nastos, J., Gao, Y.: A novel branching strategy for parameterized graph modification problems. In: Wu, W., Daescu, O. (eds.) COCOA 2010, Part II. LNCS, vol. 6509, pp. 332–346. Springer, Heidelberg (2010)
12. Niedermeier, R., Rossmanith, P.: A general method to speed up fixed-parameter tractable algorithms. *Information Processing Letters* 73, 125–129 (2000)
13. Sharan, R.: Graph modification problems and their applications to genomic research. PhD Thesis, Tel-Aviv University (2002)
14. Villanger, Y.: Proper interval vertex deletion. In: Raman, V., Saurabh, S. (eds.) IPEC 2010. LNCS, vol. 6478, pp. 228–238. Springer, Heidelberg (2010)
15. Wegner, G.: Eigenschaften der Nerven homologisch-einfacher Familien im  $R^n$ . PhD thesis, Universität Göttingen (1967)

# Parameterized Algorithms for Maximum Agreement Forest on Multiple Trees<sup>\*</sup>

Feng Shi<sup>1</sup>, Jianer Chen<sup>1,2</sup>, Qilong Feng<sup>1</sup>, and Jianxin Wang<sup>1</sup>

<sup>1</sup> School of Information Science and Engineering, Central South University, China

<sup>2</sup> Department of Computer Science and Engineering, Texas A&M University, USA

**Abstract.** The Maximum Agreement Forest problem (MAF) asks for a largest common subforest of a collection of phylogenetic trees. The MAF problem on two binary phylogenetic trees has been studied extensively in the literature. In this paper, we present the first group of fixed-parameter tractable algorithms for the MAF problem on multiple (i.e., two or more) binary phylogenetic trees. Our techniques work fine for the problem for both rooted trees and unrooted trees. The computational complexity of our algorithms is comparable with that of the known algorithms for two trees, and is independent of the number of phylogenetic trees for which a maximum agreement forest is constructed.

## 1 Introduction

Phylogenetic trees have been widely used in the study of evolutionary biology to represent the tree-like evolution of a collection of species. However, different methods often lead to different trees. In order to facilitate the comparison of different phylogenetic trees, several distance metrics have been proposed, such as Robinson-Foulds [11], NNI [10], TBR and SPR [9,13].

A graph theoretical model, the *maximum agreement forest* (MAF) of two phylogenetic trees, has been formulated for the TBR distance and the SPR distance [8] for phylogenetic trees. Define the *order* of a forest to be the number of connected components in the forest.<sup>1</sup> Allen and Steel [1] proved that the TBR distance between two unrooted binary phylogenetic trees is equal to the order of their MAF minus 1, and Bordewich and Semple [3] proved that the rSPR distance between two rooted binary phylogenetic trees is equal to the order of their rooted version of MAF minus 1. In terms of computational complexity, it is known that computing the order of an MAF is NP-hard for two unrooted binary phylogenetic trees [8], as well as for two rooted binary phylogenetic trees [3].

Thus, the order of an MAF measures the “difference” between the two phylogenetic trees constructed from the same collection of species, which can be

---

\* This work is supported by the National Natural Science Foundation of China under Grants (61103033, 61173051, 70921001), and the Doctoral Discipline Foundation of Higher Education Institution of China under Grant (20090162110056).

<sup>1</sup> The definitions for the study of maximum agreement forests have been kind of confusing. If *size* denotes the number of edges in a forest, then for a forest, the size is equal to the number of vertices minus the order. In particular, when the number of vertices is fixed, a forest of a large size means a small order of the forest.

small in practice. This observation has motivated the study of parameterized algorithms for the MAF problem, where the problem is parameterized by the order  $k$  of an MAF. A parameterized problem is *fixed-parameter tractable* [6] if it is solvable in time  $f(k)n^{O(1)}$ . In particular, for small values of the parameter  $k$ , such an algorithm may solve the problem more effectively. Allen and Steel [1] showed that the MAF problem on unrooted binary phylogenetic trees is fixed-parameter tractable. Hallett and McCartin [7] developed a faster parameterized algorithm of running time  $O(4^k k^5 + n^{O(1)})$  for the MAF problem on two unrooted binary phylogenetic trees. Whidden and Zeh [15] further improved the time complexity to  $O(4^k k + n^3)$  or  $O(4^k n)$ . A further faster algorithm has been announced recently by Chen, Fan, and Sze [5], which runs in time  $O(3^k n)$  and is currently the fastest algorithm for the MAF problem on two unrooted binary phylogenetic trees. For the MAF problem on two rooted binary phylogenetic trees, Bordewich *et al.* [2] developed a parameterized algorithm of running time  $O(4^k k^4 + n^3)$ . Whidden *et al.* [14] improved this bound and developed an algorithm of running time  $O(2.42^k k + n^3)$ . This is currently the fastest algorithm for the MAF problem on two rooted binary phylogenetic trees.

On the other hand, the computational complexity for the MAF problem on more than two phylogenetic trees has not been studied as extensively as that on two trees. Note that it makes perfect sense to investigate the MAF problem on more than two phylogenetic trees: we may construct the phylogenetic trees for the same collection of species using more than two methods. However, it seems much more difficult to construct an MAF for more than two trees than that for two trees. For example, while there have been several polynomial-time approximation algorithms of ratio 3 for the MAF problem on two rooted binary phylogenetic trees [12,14] (the same ratio even holds true for the MAF problem on two unrooted multifurcating trees [5]), the best polynomial-time approximation algorithm [4] for the MAF problem on more than two rooted binary phylogenetic trees has a ratio 8. Similarly, while there have been more than half-dozen fixed-parameter tractable algorithms for the MAF problem on two (rooted or unrooted) binary phylogenetic trees [1,2,5,7,14,15], to our best knowledge, it is still unknown whether the MAF problem on more than two (rooted or unrooted) binary phylogenetic trees is fixed-parameter tractable.

In the current paper, we will be focused on parameterized algorithms for the MAF problem on multiple (i.e., two or more) binary phylogenetic trees, for both the version of rooted trees and the version of unrooted trees. Our main contributions include an  $O(3^k n)$ -time parameterized algorithm for the MAF problem on multiple rooted binary phylogenetic trees, and an  $O(4^k n)$ -time parameterized algorithm for the MAF problem on multiple unrooted binary phylogenetic trees. Our algorithms show that these problems are fixed-parameter tractable.

Our algorithms are based on the following simple ideas that, however, require a careful and efficient implementation. Let  $\mathcal{C} = \{T_1, T_2, \dots, T_m\}$  be a collection of rooted or unrooted binary phylogenetic trees. Note that an MAF of order  $k$  for the trees in  $\mathcal{C}$  must be an agreement forest for the first two trees  $T_1$  and  $T_2$ , which although may not be necessarily maximum. Therefore, if we can essentially

examine *all* agreement forests of order bounded by  $k$  for the trees  $T_1$  and  $T_2$ , then we can easily check if any of them is an MAF for all the trees in  $\mathcal{C}$  (note that checking if a forest is a subgraph of a tree in  $\mathcal{C}$  is easy). In order to implement this idea, however, we must overcome the following difficulties. First, we must ensure that no agreement forest in our concern is missing. This in fact requires new and non-trivial techniques: *all* MAF algorithms for two trees proposed in the literature are based on resolving conflicting structures in the two trees, and do not guarantee examining all agreement forests of order bounded by  $k$ . The conflicting structures help to identify edges in the trees whose removal leads to the construction of the MAF. Therefore, if the two trees  $T_1$  and  $T_2$  do not conflict much (in an extreme case,  $T_1$  and  $T_2$  are isomorphism), then an MAF for  $T_1$  and  $T_2$  may not help much for constructing an MAF for all the trees in  $\mathcal{C}$ . Secondly, with the assurance that essentially all concerned agreement forests for  $T_1$  and  $T_2$  are examined, we must make sure that our algorithms are sufficiently efficient. This goal has also been nicely achieved: compared with the algorithms *published* in the literature, our  $O(3^k n)$ -time algorithm for the MAF problem on multiple rooted binary phylogenetic trees is asymptotically faster than the best published algorithm for the MAF problem on two rooted binary phylogenetic trees, which runs in time  $O(4^k n^{O(1)})$  [3], and our  $O(4^k n)$ -time algorithm for the MAF problem on multiple unrooted binary phylogenetic trees matches the computational complexity of the best published algorithm for the MAF problem on two unrooted binary phylogenetic trees [7]. Only very recent work on two rooted trees [14] and on two unrooted trees [5], still in the status of unpublished manuscripts, has slightly improved these bounds, which, however, do not seem to be extendable to the problems on more than two trees. On the other hand, our algorithms work fine for the MAF problems for an arbitrary number of trees.

## 2 Definitions and Problem Formulations

A tree is a *single-vertex tree* if it consists of a single vertex, which is the leaf of the tree. A tree is a *single-edge tree* if it consists of a single edge. A tree is *binary* if either it is a single-vertex tree or each of its vertices has degree either 1 or 3. The degree-1 vertices are *leaves* and the degree-3 vertices are *non-leaves* of the tree. There are two versions in our discussion, one is on unrooted trees and the other is on rooted trees. We first give the terminologies on the unrooted version, then remark on the differences for the rooted version. Let  $X$  be a fixed *label-set*.

### Unrooted $X$ -Trees and $X$ -Forests

A binary tree is *unrooted* if no root is specified in the tree – in this case no ancestor-descendant relation is defined in the tree. For the label-set  $X$ , an unrooted *binary phylogenetic  $X$ -tree*, or simply an unrooted  $X$ -tree, is an unrooted binary tree whose leaves are labeled bijectively by the label-set  $X$  (all non-leaves are not labeled). An unrooted  $X$ -tree will also be called an (unrooted) *leaf-labeled tree* if the label-set  $X$  is irrelevant. A *subforest* of an unrooted  $X$ -tree  $T$  is a subgraph of  $T$ , and a *subtree* of  $T$  is a connected subgraph of  $T$ . An unrooted  $X$ -forest  $F$  is a subforest of an unrooted  $X$ -tree  $T$  that contains all

leaves of  $T$  such that each connected component of  $F$  contains at least one leaf in  $T$ . Thus, an unrooted  $X$ -forest  $F$  is a collection of leaf-labeled trees whose label-sets are disjoint such that the union of the label-sets is equal to  $X$ . Define the *order* of the  $X$ -forest  $F$ , denoted  $\text{Ord}(F)$ , to be the number of connected components in  $F$ . For a subset  $X'$  of the label-set  $X$ , the *subtree induced by  $X'$*  in an unrooted  $X$ -tree  $T$ , denoted by  $T[X']$ , is the minimal subtree of  $T$  that contains all leaves with labels in  $X'$ .

A subtree  $T'$  of an unrooted  $X$ -tree may contain unlabeled vertices of degree less than 3. In this case we apply the *forced contraction* operation on  $T'$ , which replaces each degree-2 vertex  $v$  and its incident edges with a single edge connecting the two neighbors of  $v$ , and removes each unlabeled vertex that has degree smaller than 2. Note that the forced contraction does not change the order of an  $X$ -forest. An  $X$ -forest  $F$  is *strongly reduced* if the forced contraction does not apply to  $F$ . It has been well-known that the forced contraction operation does not affect the construction of an MAF for  $X$ -trees (see, for example, [2,7]). Therefore, we will assume that the forced contraction is applied immediately whenever it is applicable. Thus, the  $X$ -forests in our discussion are always assumed to be strongly reduced. With this assumption, a unlabeled vertex in an unrooted  $X$ -trees is always of degree 3. If a leaf-labeled forest  $F'$  is isomorphic to a subforest of an  $X$ -forest  $F$  (up to the forced contraction), then we will simply say that  $F'$  is a subforest of  $F$ .

### **Rooted $X$ -Trees and $X$ -Forests**

A binary tree is *rooted* if a particular leaf is designated as the root (so it is *both* a root and a leaf), which specifies a unique ancestor-descendant relation in the tree. A rooted  $X$ -tree is a rooted binary tree whose leaves are labeled bijectively by the label-set  $X$ . The root of an  $X$ -tree will always be labeled by a special symbol  $\rho$  in  $X$ . A subtree  $T'$  of a rooted  $X$ -tree  $T$  is a connected subgraph of  $T$  which contains at least one leaf in  $T$ . In order to preserve the ancestor-descendant relation in  $T$ , we should define the root of the subtree of  $T$ . If  $T'$  contains the leaf  $\rho$ , certainly, it is the root of the subtree; if  $T'$  does not contain the leaf  $\rho$ , the node in  $T'$  which is the least common ancestor of the leaves in  $T'$  is defined to be the root of  $T'$ . A subforest of a rooted  $X$ -tree  $T$  is defined to be a subgraph of  $T$ . A (rooted)  $X$ -forest  $F$  is a subforest of a rooted  $X$ -tree  $T$  that contains a collection of subtrees whose label-sets are disjoint such that the union of the label-sets is equal to  $X$ . Thus, one of the subtrees in a rooted  $X$ -forest  $F$  must have the vertex labeled  $\rho$  as its root.

We again assume that the forced contraction is applied immediately whenever it is applicable. However, if the root  $r$  of a subtree  $T'$  is of degree 2, then the operation will *not* be applied on  $r$ , in order to preserve the ancestor-descendant relation in  $T$ . Therefore, after the forced contraction, the root of a subtree  $T'$  of a rooted  $X$ -tree is either an unlabeled vertex of degree-2, or the vertex labeled  $\rho$  of degree-1, or a labeled vertex of degree-0. All unlabeled vertices in  $T'$  that is not the root of  $T'$  have degree 3. We say that a leaf-labeled forest  $F'$  is a subforest of a rooted  $X$ -forest  $F$  if  $F'$  is isomorphic to a subforest of the  $X$ -forest  $F$  (up to the forced contraction).

## Agreement Forests

The following terminologies are used for both rooted and unrooted versions.

An  $X$ -forest  $F$  is an *agreement forest* for a collection  $\{F_1, F_2, \dots, F_m\}$  of  $X$ -forests if  $F$  is a subforest of  $F_i$ , for all  $i$ . A *maximum agreement forest* (abbr. *MAF*)  $F^*$  for  $\{F_1, F_2, \dots, F_m\}$  is an agreement forest for  $\{F_1, F_2, \dots, F_m\}$  with a minimum  $\text{Ord}(F^*)$  over all agreement forests for  $\{F_1, F_2, \dots, F_m\}$ .

The problems we are focused on are parameterized versions of the Maximum Agreement Forest Problem for an arbitrary number of  $X$ -trees, with a rooted version and an unrooted version, which are formally given as follows.

### ROOTED MAXIMUM AGREEMENT FOREST (rooted-MAF)

*Input:* A set  $\{F_1, \dots, F_m\}$  of rooted  $X$ -forests, and a parameter  $k$

*Output:* an agreement forest  $F^*$  for  $\{F_1, \dots, F_m\}$  with  $\text{Ord}(F^*) \leq k$ ,  
or report that no such an agreement forest exists

### UNROOTED MAXIMUM AGREEMENT FOREST (unrooted-MAF)

*Input:* A set  $\{F_1, \dots, F_m\}$  of unrooted  $X$ -forests, and a parameter  $k$

*Output:* an agreement forest  $F^*$  for  $\{F_1, \dots, F_m\}$  with  $\text{Ord}(F^*) \leq k$ ,  
or report that no such an agreement forest exists

When each of the  $X$ -forests  $F_1, \dots, F_m$  is an  $X$ -tree, the above problems become the standard Maximum Agreement Forest Problems on multiple binary phylogenetic trees, for the rooted version and the unrooted version, respectively.

The following concept on two  $X$ -forests will be important in our discussion, which applies to both rooted version and the unrooted version.

**Definition 1.** Let  $F_1$  and  $F_2$  be two  $X$ -forests (either both rooted or both unrooted). An agreement forest  $F$  for  $F_1$  and  $F_2$  is a *maximal agreement forest* (*maximal-AF*) for  $F_1$  and  $F_2$  if there is no agreement forest  $F'$  for  $F_1$  and  $F_2$  such that  $F$  is a subforest of  $F'$  and  $\text{Ord}(F') < \text{Ord}(F)$ .

By definition, an MAF for two  $X$ -forests  $F_1$  and  $F_2$  is also a maximal-AF for  $F_1$  and  $F_2$ . Note that *every* agreement forest for two  $X$ -forests  $F_1$  and  $F_2$  is a subforest of a maximal-AF  $F'$  for  $F_1$  and  $F_2$ , but  $F'$  may not be unique.

## 3 Maximal-AF for Two $X$ -Forests

Fix a label-set  $X$ . Because of the bijection between the leaves in an  $X$ -forest  $F$  and the elements in the label-set  $X$ , sometimes we will use, without confusion, a label in  $X$  to refer to the corresponding leaf in  $F$ , or vice versa.

Let  $F_1$  and  $F_2$  be two  $X$ -forests, either both are rooted or both are unrooted. In this section, we discuss how we enumerate *all* maximal-AF for  $F_1$  and  $F_2$ . The discussion is divided into the case for the rooted version and the case for the unrooted version.

### Rooted Maximal-AF

In this case, both  $F_1$  and  $F_2$  are rooted  $X$ -forests. We proceed by repeatedly removing edges in  $F_1$  and  $F_2$  until certain condition is met. Let  $F^*$  be a fixed maximal-AF for  $F_1$  and  $F_2$ .

Two labels  $a$  and  $b$  (and their corresponding leaves) in a forest are *siblings* if they have the common parent. We start with the following simple lemma.

**Lemma 1.** Let  $F_1$  and  $F_2$  be two strongly reduced rooted  $X$ -forests. If  $F_2$  contains no sibling pairs, then  $F_1$  and  $F_2$  has a unique maximal-AF that can be constructed in linear time.

*Proof.* Let  $T$  be a connected component of  $F_2$ , which is a rooted leaf-labeled tree. If  $F_2$  contains no sibling pairs, then neither does  $T$ . Therefore, if  $T$  does not contain the root  $\rho$ , then  $T$  must be a single-vertex tree whose leaf is a labeled vertex. If  $T$  contains  $\rho$ , then  $T$  is either a single-vertex tree whose leaf is  $\rho$  or a single-edge tree whose root is  $\rho$  with a unique child that is labeled by a label  $\tau$ . Thus, all connected components of the  $X$ -forest  $F_2$  are single-vertex trees, except at most one that is a single-edge tree whose two leaves are labeled by the elements  $\rho$  and  $\tau$  in  $X$ . Therefore, if the leaves  $\rho$  and  $\tau$  are in the same connected component in the  $X$ -forest  $F_1$ , then the (unique) maximal-AF for  $F_1$  and  $F_2$  is the  $X$ -forest  $F_2$  itself. On the other hand, if  $\rho$  and  $\tau$  are in different connected components in  $F_1$ , then the maximal-AF (again unique) for  $F_1$  and  $F_2$  consists of only single-vertex trees, each is labeled by an element in  $X$ .  $\square$

By Lemma 1, therefore, in the following discussion, we will assume that the rooted  $X$ -forest  $F_2$  has a sibling pair  $(a, b)$ . By definition,  $a$  and  $b$  cannot be  $\rho$ . Let  $p_2$ , which is an unlabeled vertex, be the parent of  $a$  and  $b$  in  $F_2$ . If one of  $a$  and  $b$  is a single-vertex tree in the  $X$ -forest  $F_1$ , then we can remove the edge in  $F_2$  that is incident to the label, and break up the sibling pair in  $F_2$ . Thus, in the following discussion, we assume that none of  $a$  and  $b$  is a single-vertex tree in  $F_1$ . Let  $p_1$  and  $p'_1$  be the parents of  $a$  and  $b$  in  $F_1$ , respectively. We consider all possible cases for the labels  $a$  and  $b$  in the  $X$ -forest  $F_1$ .

**Case 1.** The labels  $a$  and  $b$  are in different connected components in  $F_1$ .

In this case,  $a$  and  $b$  cannot be in the same connected component in the maximal-AF  $F^*$ . Therefore, one of the edges  $[a, p_2]$  and  $[b, p_2]$  in  $F_2$  must be removed, which forces one of the labels  $a$  and  $b$  to be a single-vertex tree in the maximal-AF  $F^*$ . Therefore, in this case, we apply the following branching step:

- Step 1.** (branch-1) remove the edge  $[a, p_1]$  in  $F_1$  and the edge  $[a, p_2]$  in  $F_2$   
to make  $a$  a single-vertex tree in both  $F_1$  and  $F_2$ ;
- (branch-2) remove the edge  $[b, p'_1]$  in  $F_1$  and the edge  $[b, p_2]$  in  $F_2$   
to make  $b$  a single-vertex tree in both  $F_1$  and  $F_2$ .

One of these branches will keep  $F^*$  a maximal-AF for the new  $F_1$  and  $F_2$ .

**Case 2.** The labels  $a$  and  $b$  are also siblings in  $F_1$ , i.e.,  $p_1 = p'_1$ .

Since  $F^*$  is a maximal-AF, in this case,  $a$  and  $b$  must be also siblings in  $F^*$ . Therefore, the structure that consists of  $a$  and  $b$  and their parent remains unchanged when we construct  $F^*$  from  $F_1$  and  $F_2$  by removing edges in  $F_1$  and  $F_2$ . Thus, this structure can be regarded as a single leaf labeled by a “combined” label  $\underline{ab}$  in both  $F_1$  and  $F_2$ . To implement this, we apply the following step:

**Step 2.** Remove  $a$  and  $b$ , and make their parent a new leaf labeled  $\underline{ab}$ , in both  $F_1$  and  $F_2$ .

We call the operation in Step 2 “shrinking  $a$  and  $b$  into a new label  $\underline{ab}$ ”. This step not only changes the structure of  $F_1$  and  $F_2$ , but also replaces the label-set

$X$  with a new label-set  $(X \setminus \{a, b\}) \cup \{ab\}$ . If we also apply this operation in the maximal-AF  $F^*$ , then the new  $F^*$  remains a maximal-AF for  $F_1$  and  $F_2$ .

**Case 3.** The labels  $a$  and  $b$  are in the same connected component in  $F_1$  but are not siblings.

Let  $P = \{a, c_1, c_2, \dots, c_r, b\}$  be the unique path in  $F_1$  connecting  $a$  and  $b$ , in which  $c_h$  is the least common ancestor of  $a$  and  $b$ ,  $1 \leq h \leq r$ . Since  $a$  and  $b$  are not siblings,  $r \geq 2$ . See Figure 1(a) for an illustration. There are three subcases.

SUBCASE 3.1.  $a$  is a single-vertex tree in  $F^*$ . Then removing the edge incident to  $a$  in both  $F_1$  and  $F_2$  keeps  $F^*$  a maximal-AF for  $F_1$  and  $F_2$ .

SUBCASE 3.2.  $b$  is a single-vertex tree in  $F^*$ . Then removing the edge incident to  $b$  in both  $F_1$  and  $F_2$  keeps  $F^*$  a maximal-AF for  $F_1$  and  $F_2$ .

SUBCASE 3.3. Neither of  $a$  and  $b$  is a single-vertex tree in  $F^*$ . Then the two edges that are incident to  $a$  and  $b$  in  $F_2$  must be kept in  $F^*$ . Therefore,  $a$  and  $b$  are siblings in  $F^*$ . On the other hand, in order to make  $a$  and  $b$  siblings in the  $X$ -forest  $F_1$ , all edges that are not on the path  $P$  but are incident to a vertex  $c_j$  in  $P$ , where  $j \neq h$ , must be removed (note that this is because the subtrees in an  $X$ -forest must preserve the ancestor-descendant relation). Note that since  $r \geq 2$ , there is at least one such an edge. Therefore, in this subcase, if we remove all these edges, then  $F^*$  remains a maximal-AF for  $F_1$  and  $F_2$ .

Summarizing the above analysis, in Case 3, we apply the following step:

- Step 3.** (branch-1) remove the edge incident to  $a$  in both  $F_1$  and  $F_2$ ;  
 (branch-2) remove the edge incident to  $b$  in both  $F_1$  and  $F_2$ ;  
 (branch-3) remove all edges in  $F_1$  that are not on the path  $P$  connecting  $a$  and  $b$  but are incident to a vertex in  $P$ , except the one that is incident to the least common ancestor of  $a$  and  $b$ .

One of these branches must keep  $F^*$  a maximal-AF for the new  $F_1$  and  $F_2$ .

Therefore, for two given rooted  $X$ -forests  $F_1$  and  $F_2$ , if we iteratively apply the above process, branching accordingly based on the cases, then the process will end up with a pair  $(F_1, F_2)$  in which  $F_2$  contains no sibling pairs. When this occurs, the process applies the following step:

**Final Step.** if  $F_2$  has no sibling pairs, then construct the maximal-AF  $F^*$  for  $F_1$  and  $F_2$ , and convert  $F^*$  into an agreement forest for the original  $F_1$  and  $F_2$ .

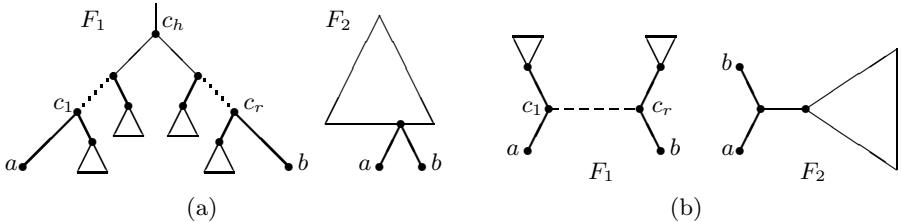
When  $F_2$  contains no sibling pairs, by Lemma 1, we can construct the (unique) maximal-AF  $F^*$  for  $F_1$  and  $F_2$  in linear time. The forest  $F^*$  may not be a subforest of the original  $F_1$  and  $F_2$  because Step 2 shrinks labels. For this, we should “expand” the shrunk labels, in a straightforward way. Note that this expanding process may be applied iteratively.

Summarizing the above discussion, we conclude with the following lemma.

**Lemma 2.** Let  $F_1$  and  $F_2$  be two rooted  $X$ -forests. If we apply Steps 1-3 iteratively until  $F_2$  contains no sibling pairs, then for every maximal-AF  $F^*$  for the original  $F_1$  and  $F_2$ , at least one of the branches in the process produces the maximal-AF  $F^*$  in its Final Step.

*Proof.* Fix a maximal-AF  $F^*$  for  $F_1$  and  $F_2$ . By the above analysis, for each of the cases, at least one of the branches in the corresponding step keeps  $F^*$

a maximal-AF for  $F_1$  and  $F_2$ . Moreover, when  $F_2$  contains no sibling pairs, the maximal-AF for  $F_1$  and  $F_2$  becomes unique. Combining these two facts, we conclude that at least one of the branches in the process ends up with an pair  $F_1$  and  $F_2$  whose maximal-AF, after the final step, is  $F^*$ . Since  $F^*$  is an arbitrary maximal-AF for  $F_1$  and  $F_2$ , the lemma is proved.  $\square$



**Fig. 1.** The path connecting the labels  $a$  and  $b$  in  $F_1$  when  $F_1$  is (a) rooted; (b) unrooted

### Unrooted Maximal-AF

The analysis for the unrooted version proceeds in a similar manner. However, since an unrooted tree enforces no ancestor-descendant relation in the tree, subtrees in the tree have no requirement of preserving such a relation. This fact induces certain subtle differences.

Let  $F_1$  and  $F_2$  be two unrooted  $X$ -forests, and let  $F^*$  be a fixed maximal-AF for  $F_1$  and  $F_2$ . Recall that we assume  $F_1$  and  $F_2$  to be strongly reduced.

Two labels  $a$  and  $b$  in an unrooted  $X$ -forest  $F$  are *siblings* if either they are the two leaves of a single-edge tree in  $F$ , or they are adjacent to the same non-leaf vertex in  $F$ , which will be called the “parent” of  $a$  and  $b$ .

An unrooted  $X$ -forest with no sibling pairs has an even simpler structure: all its connected components are single-vertex trees. Thus, we again have:

**Lemma 3.** *Let  $F_1$  and  $F_2$  be two unrooted  $X$ -forests. If  $F_2$  contains no sibling pairs, then the maximal-AF for  $F_1$  and  $F_2$  can be constructed in linear time.*

Thus, again we will assume that the unrooted  $X$ -forest  $F_2$  has a sibling pair  $(a, b)$ . Also we can assume that none of  $a$  and  $b$  is a single-vertex tree in  $F_1$ .

**Case 1.** The labels  $a$  and  $b$  are in different connected components in  $F_1$ .

In this case, again one of the labels  $a$  and  $b$  must be a single-vertex tree in the maximal-AF  $F^*$ . Therefore, we apply the following step:

**Step 1.** (branch-1) remove the edge incident to  $a$  in both  $F_1$  and  $F_2$  to make

$a$  a single-vertex tree in both  $F_1$  and  $F_2$ ;

(branch-2) remove the edge incident to  $b$  in both  $F_1$  and  $F_2$  to make

$b$  a single-vertex tree in both  $F_1$  and  $F_2$ .

**Case 2.** The labels  $a$  and  $b$  are also siblings in  $F_1$ .

We have to be a bit more careful for this case since a sibling pair may come from a single-edge tree. There are three different cases: (1)  $a$  and  $b$  come from a single-edge tree in both  $F_1$  and  $F_2$ ; (2)  $a$  and  $b$  come from a single-edge tree in exact one of  $F_1$  and  $F_2$ ; and (3)  $a$  and  $b$  have a common parent in both  $F_1$  and  $F_2$ . By a careful analysis and noticing that  $F^*$  is maximal, we can verify that in

all these subcases it is always safe to shrink  $a$  and  $b$  into a new label, which is implemented by the following step:

**Step 2.** Shrink the labels  $a$  and  $b$  in both  $F_1$  and  $F_2$ : if  $a$  and  $b$  have a common parent, then remove the edges incident to  $a$  and  $b$  and make their parent a new leaf labeled  $\underline{ab}$ ; if  $a$  and  $b$  come from a single-edge tree, then combine them into a single vertex labeled  $\underline{ab}$ .

After this process, the maximal-AF  $F^*$  for  $F_1$  and  $F_2$ , in which the labels  $a$  and  $b$  are also shrunk, remains a maximal-AF forest for the new  $F_1$  and  $F_2$ .

**Case 3.**  $a$  and  $b$  are in the same connected component in  $F_1$  but are not siblings.

Let  $P = \{a, c_1, c_2, \dots, c_r, b\}$  be the unique path in  $F_1$  that connects  $a$  and  $b$ , where  $r \geq 2$ . See Figure 1(b) for an illustration. The cases in which either  $a$  or  $b$  is a single-vertex tree in  $F^*$  again cause removing the edge incident to  $a$  or  $b$  in  $F_1$ . However, when  $a$  and  $b$  are siblings in  $F^*$ , then in  $F_1$ , at most one of the edges that are not on the path  $P$  but are incident to a vertex in  $P$  can be kept. However, since the subtree in an unrooted forest does not need to preserve any ancestor-descendant relation, we cannot decide which of these edges should be kept. On the other hand, since  $r \geq 2$ , we know at least one of the two edges, which are not on the path  $P$  but are incident to  $c_1$  and  $c_r$ , respectively, must be removed. Therefore, we can branch by removing either the one incident to  $c_1$  or the one incident to  $c_r$ . In summary, in Case 3, we apply the following step:

- Step 3.** (branch-1) remove the edge incident to  $a$  in both  $F_1$  and  $F_2$ ;  
 (branch-2) remove the edge incident to  $b$  in both  $F_1$  and  $F_2$ ;  
 (branch-3) remove the edge incident to  $c_1$  but not on the path  $P$  in  $F_1$ ;  
 (branch-4) remove the edge incident to  $c_r$  but not on the path  $P$  in  $F_1$ .

One of these branches must keep  $F^*$  a maximal-AF for the new  $F_1$  and  $F_2$ .

Again if the unrooted  $X$ -forest  $F_2$  contains no sibling pairs, then we apply Lemma 3 to construct the maximal-AF for  $F_1$  and  $F_2$  by the following step:

**Final Step.** If  $F_2$  contains no sibling pairs, then construct the maximal-AF  $F^*$  for  $F_1$  and  $F_2$ , and convert  $F^*$  into an agreement forest for the original  $F_1$ ,  $F_2$ .

The above analysis finally gives the following conclusion, whose proof is exactly the same as that of Lemma 2 for the rooted version.

**Lemma 4.** *Let  $F_1$  and  $F_2$  be two unrooted  $X$ -forests. If we apply Steps 1-3 iteratively until  $F_2$  contains no sibling pairs, then for every maximal-AF  $F^*$  for the original  $F_1$  and  $F_2$ , at least one of the branches in the process produces the maximal-AF  $F^*$  in its Final Step.*

## 4 The Parameterized Algorithms

Now we are ready for presenting the parameterized algorithms for the MAF problem, for both the rooted version as well as the unrooted version. Let  $F_1, F_2, \dots, F_m$  be  $m$   $X$ -forests, either all are rooted or all are unrooted. We first give a few lemmas, which hold true for both rooted and unrooted versions. Assume  $m \geq 3$ .

The first lemma follows directly from the definition.

**Lemma 5.** Let  $F'$  be an agreement forest for  $F_1$  and  $F_2$ . Then every agreement forest for  $\{F', F_3, \dots, F_m\}$  is an agreement forest for  $\{F_1, F_2, \dots, F_m\}$ . If  $F'$  contains an MAF for  $\{F_1, F_2, \dots, F_m\}$ , then an MAF for  $\{F', F_3, \dots, F_m\}$  is also an MAF for  $\{F_1, F_2, \dots, F_m\}$ .

**Lemma 6.** For every MAF  $F$  for  $\{F_1, F_2, \dots, F_m\}$ , there is a maximal-AF  $F^*$  for  $F_1$  and  $F_2$  such that  $F$  is also an MAF for  $\{F^*, F_3, \dots, F_m\}$ .

*Proof.* Let  $F_0$  be an MAF for  $\{F_1, F_2, \dots, F_m\}$ . Then  $F_0$  is an agreement forest for  $F_1$  and  $F_2$ . Let  $F^*$  be a maximal-AF for  $F_1$  and  $F_2$  that has  $F_0$  as a subforest. Then  $F_0$  is an agreement forest for  $\{F^*, F_3, \dots, F_m\}$ . Therefore, the order of an MAF for  $\{F^*, F_3, \dots, F_m\}$  is at most  $\text{Ord}(F_0)$ . On the other hand, since  $F^*$  is a subforest of both  $F_1$  and  $F_2$ , every agreement forest for  $\{F^*, F_3, \dots, F_m\}$  is also an agreement forest for  $\{F_1, F_2, \dots, F_m\}$ . Therefore, the order of an MAF for  $\{F^*, F_3, \dots, F_m\}$  is at least  $\text{Ord}(F_0)$ , thus must be equal to  $\text{Ord}(F_0)$ . Since  $F_0$  is an agreement forest for  $\{F^*, F_3, \dots, F_m\}$ ,  $F_0$  must be an MAF for  $\{F^*, F_3, \dots, F_m\}$ .  $\square$

Now consider an instance  $(F_1, F_2, \dots, F_m; k)$  of MAF, either rooted or unrooted. For a subforest  $F'$  of a forest  $F$ , we always have  $\text{Ord}(F) \leq \text{Ord}(F')$ . Thus, no maximal-AF  $F$  for  $F_1$  and  $F_2$  with  $\text{Ord}(F) > k$  can contain an MAF  $F'$  for  $(F_1, F_2, \dots, F_m)$  with  $\text{Ord}(F') \leq k$ , so we only need to examine all maximal-AFs whose order is bounded by  $k$ . An outline of our algorithm works as follows:

#### Main-Algorithm

1. construct a collection  $\mathcal{C}$  of agreement forests for  $F_1$  and  $F_2$  that contains all maximal-AF  $F^*$  for  $F_1$  and  $F_2$  with  $\text{Ord}(F^*) \leq k$ ;
2. **for** each agreement forest  $F$  for  $F_1$  and  $F_2$  constructed in step 1 **do**  
    recursively work on the instance  $(F, F_3, \dots, F_m; k)$ .

**Theorem 1.** The Main-Algorithm correctly returns an agreement forest  $F$  for  $\{F_1, F_2, \dots, F_m\}$  with  $\text{Ord}(F) \leq k$  if such an agreement forest exists.

*Proof.* For an  $F'$  in the collection  $\mathcal{C}$ , by Lemma 5, a solution to  $(F', F_3, \dots, F_m; k)$  returned by step 2 is also a solution to  $(F_1, F_2, \dots, F_m; k)$ . On the other hand, if  $(F_1, F_2, \dots, F_m; k)$  has a solution, then an MAF  $F_0$  for  $\{F_1, F_2, \dots, F_m\}$  satisfies  $\text{Ord}(F_0) \leq k$ . For the maximal-AF  $F^*$  for  $F_1$  and  $F_2$  that contains  $F_0$ , by Lemma 6,  $F_0$  is also a solution to  $(F^*, F_3, \dots, F_m; k)$ , which is an instance examined in step 2. On this instance, Step 2 will return a solution that, by Lemma 5, is also a solution to  $(F_1, F_2, \dots, F_m; k)$ .  $\square$

In the following, we present the details for the Main-Algorithm for the rooted version. By Theorem 1, our must carefully check that all maximal-AF's  $F$  for  $F_1$  and  $F_2$  with  $\text{Ord}(F) \leq k$  be constructed in the collection  $\mathcal{C}$ . Also, we should develop algorithms to achieve the desired complexity bounds.

#### A Parameterized Algorithm for Rooted-MAF

The parameterized algorithm for rooted-MAF is a combination of the analysis given in Section 3 and the Main-Algorithm, which is given in Figure 2.

**Algorithm.** Rt-MAF( $F_1, F_2, \dots, F_m; k$ )

Input: a collection  $\{F_1, F_2, \dots, F_m\}$  of rooted  $X$ -forests,  $m \geq 1$ , and a parameter  $k$   
 Output: an agreement forest  $F^*$  for  $\{F_1, F_2, \dots, F_m\}$  with  $\text{Ord}(F^*) \leq k$  if  $F^*$  exists

1. if ( $m = 1$ ) then if ( $\text{Ord}(F_1) \leq k$ ) then return  $F_1$  else return('no');
2. if ( $\text{Ord}(F_1) > k$ ) then return('no');
3. if a label  $a$  is a single-vertex tree in exactly one of  $F_1$  and  $F_2$   
 then make  $a$  a single-vertex tree in both  $F_1$  and  $F_2$ ;
4. if  $F_2$  has no sibling pairs  
 then let  $F'$  be the maximal-AF for  $F_1$  and  $F_2$ ; return Rt-MAF( $F', F_3, \dots, F_m; k$ );
5. let  $(a, b)$  be a sibling pair in  $F_2$ ;
6. if  $a$  and  $b$  are in different connected components in  $F_1$   
 then branch:
  1. make  $a$  a single-vertex tree in both  $F_1$  and  $F_2$ ; return Rt-MAF( $F_1, F_2, \dots, F_m; k$ );
  2. make  $b$  a single-vertex tree in both  $F_1$  and  $F_2$ ; return Rt-MAF( $F_1, F_2, \dots, F_m; k$ );
7. if  $a$  and  $b$  are also siblings in  $F_1$   
 then shrink  $a, b$  into a new leaf  $\underline{ab}$  in  $F_1$  and  $F_2$ ; return Rt-MAF( $F_1, F_2, \dots, F_m; k$ );
8. let  $P = \{a, c_1, \dots, c_r, b\}$  be the unique path in  $F_1$  connecting  $a$  and  $b$ ,  $r \geq 2$ ;  
 then branch:
  1. make  $a$  a single-vertex tree in both  $F_1$  and  $F_2$ ; return Rt-MAF( $F_1, F_2, \dots, F_m; k$ );
  2. make  $b$  a single-vertex tree in both  $F_1$  and  $F_2$ ; return Rt-MAF( $F_1, F_2, \dots, F_m; k$ );
  3. remove all edges in  $F_1$  not on  $P$  but incident to a vertex in  $P$ , except the one  
 incident to the least common ancestor of  $a, b$ ; return Rt-MAF( $F_1, F_2, \dots, F_m; k$ ).

**Fig. 2.** Algorithm for the Rooted-MAF problem

The algorithm is a branch-and-search process. Its execution can be depicted by a search tree  $\mathcal{T}$  whose leaves correspond to conclusions or solutions generated by the algorithm based on different branches. Each internal node of the search tree  $\mathcal{T}$  corresponds to a branch in the search process at Steps 6 or 8 based on an instance of the problem. We call a path from the root to a leaf in the search tree  $\mathcal{T}$  a *computational path* in the process. The algorithm returns an agreement forest for the original input if and only if there is a computational path that outputs the forest.

The correctness and complexity of the algorithm can be verified based on the corresponding search tree  $\mathcal{T}$ . Due to the space limit, here we just give the concluding theorem, the entire discussion for this case will be given in a complete version.

**Theorem 2.** *The rooted-MAF problem can be solved in time  $O(3^k n)$ .*

### A Parameterized Algorithm for Unrooted-MAF

The parameterized algorithm for unrooted-MAF proceeds in a similar way, based on the corresponding analysis given in Section 3. Due to the space limit, we only present its main result below, the specific algorithm for unrooted-MAF and the entire discussion for this case will be given in the complete version.

**Theorem 3.** *The unrooted-MAF problem can be solved in time  $O(4^k n)$ .*

## 5 Conclusion

In this paper, we presented two parameterized algorithms for the Maximum Agreement Forest problem on multiple binary phylogenetic trees: one for rooted trees that runs in time  $O(3^k n)$ , and the other for unrooted trees that runs in time  $O(4^k n)$ . To our best knowledge, these are the first group of fixed-parameter tractable algorithms for the Maximum Agreement Forest problem on multiple phylogenetic trees. Further improvements on the algorithm complexity are certainly desired to make the algorithms more practical in their applications. On the other hand, such an improvement seems to require new observations and new algorithmic techniques: the complexity of our algorithms for multiple phylogenetic trees is not much worse than that of the known algorithms for two phylogenetic trees – some of these algorithms were just developed very recently.

## References

1. Allen, B., Steel, M.: Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of Combinatorics* 5(1), 1–15 (2001)
2. Bordewich, M., McCartin, C., Semple, C.: A 3-approximation algorithm for the subtree distance between phylogenies. *J. Discrete Algorithms* 6(3), 458–471 (2008)
3. Bordewich, M., Semple, C.: On the computational complexity of the rooted subtree prune and regraft distance. *Annals of Combinatorics* 8(4), 409–423 (2005)
4. Chataigner, F.: Approximating the maximum agreement forest on  $k$  trees. *Information Processing Letters* 93, 239–244 (2005)
5. Chen, J., Fan, J.-H., Sze, S.-H.: Improved algorithms for the maximum agreement forest problem on general trees. In: WG 2013 (submitted, 2013)
6. Downey, R., Fellows, M.: Parameterized Complexity. Springer, New York (1999)
7. Hallett, M., McCartin, C.: A faster FPT algorithm for the maximum agreement forest problem. *Theory of Computing Systems* 41(3), 539–550 (2007)
8. Hein, J., Jiang, T., Wang, L., Zhang, K.: On the complexity of comparing evolutionary trees. *Discrete Applied Mathematics* 71, 153–169 (1996)
9. Hodson, F., Kendall, D., Tauta, P. (eds.): The recovery of trees from measures of dissimilarity. Mathematics in the Archaeological and Historical Sciences, pp. 387–395. Edinburgh University Press, Edinburgh (1971)
10. Li, M., Tromp, J., Zhang, L.: On the nearest neighbour interchange distance between evolutionary trees. *Journal on Theoretical Biology* 182(4), 463–467 (1996)
11. Robinson, D., Foulds, L.: Comparison of phylogenetic trees. *Mathematical Biosciences* 53(1-2), 131–147 (1981)
12. Rodrigues, E., Sagot, M., Wakabayashi, Y.: The maximum agreement forest problem: approximation algorithms and computational experiments. *Theoretical Computer Science* 374(1-3), 91–110 (2007)
13. Swofford, D., Olsen, G., Waddell, P., Hillis, D.: Phylogenetic inference. In: Molecular Systematics, 2nd edn., pp. 407–513. Sinauer Associates (1996)
14. Whidden, C., Beiko, R., Zeh, N.: Fixed-parameter and approximation algorithms for maximum agreement forests. CoRR. abs/1108.2664 (2011)
15. Whidden, C., Zeh, N.: A unifying view on approximation and FPT of agreement forests. In: Salzberg, S.L., Warnow, T. (eds.) WABI 2009. LNCS, vol. 5724, pp. 390–402. Springer, Heidelberg (2009)

# Small $H$ -Coloring Problems for Bounded Degree Digraphs

Pavol Hell<sup>1,\*</sup> and Auroshish Mishra<sup>2,\*\*</sup>

<sup>1</sup> School of Computing Science, Simon Fraser University, Canada - V5A 1S6  
[pavol@sfu.ca](mailto:pavol@sfu.ca)

<sup>2</sup> Department of Computer Science, Cornell University, USA - 14853  
[auroshish@cs.cornell.edu](mailto:auroshish@cs.cornell.edu)

**Abstract.** An NP-complete coloring or homomorphism problem may become polynomial time solvable when restricted to graphs with degrees bounded by a small number, but remain NP-complete if the bound is higher. For instance, 3-colorability of graphs with degrees bounded by 3 can be decided by Brooks' theorem, while for graphs with degrees bounded by 4, the 3-colorability problem is NP-complete. We investigate an analogous phenomenon for digraphs, focusing on the three smallest digraphs  $H$  with NP-complete  $H$ -colorability problems. It turns out that in all three cases the  $H$ -coloring problem is polynomial time solvable for digraphs with in-degrees at most 1, regardless of the out-degree bound (respectively with out-degrees at most 1, regardless of the in-degree bound). On the other hand, as soon as both in- and out-degrees are bounded by constants greater than or equal to 2, all three problems are again NP-complete. A conjecture proposed for graphs  $H$  by Feder, Hell and Huang states that any variant of the  $H$ -coloring problem which is NP-complete without degree constraints is also NP-complete with degree constraints, provided the degree bounds are high enough. Thus, our results verify the conjecture with very precise bounds on both in- and out-degrees that are needed for NP-completeness; in particular, the bounds underscore the fact that the sufficiently large bound must apply to both the in-degrees and the out-degrees.

## 1 Introduction

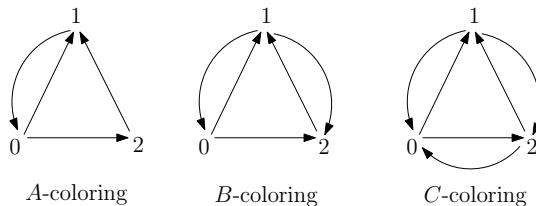
Graph coloring problems arise naturally in several contexts of both theoretical and applied nature. Be it scheduling events, solving pattern matching problems, or allocating registers to processes, many constraint satisfaction problems can be easily modeled in the graph coloring setting. A slightly more general problem is that of  $H$ -coloring.

Consider a fixed graph  $H$ . A *homomorphism*  $f : G \rightarrow H$  is a mapping  $f : V(G) \rightarrow V(H)$  such that  $f(u)f(v)$  is an edge of  $H$  for each edge  $uv$  of  $G$ . An

---

\* Partially supported by NSERC (Canada) and by ERCCZ LL 1201 Cores (Czech Republic).

\*\* Supported by the MITACS Globalink Internship Program for undergraduate students.

**Fig. 1.** The digraphs  $A$ ,  $B$ , and  $C$ 

$H$ -coloring of  $G$  is a homomorphism  $G \rightarrow H$ . The definition is formally the same for digraphs  $G, H$ : since the edges are directed, the homomorphism  $f$  is a mapping that preserves both the edges and their direction.

The  $H$ -coloring problem is the decision problem which asks the following question:

*Given a graph  $G$ , is it possible to  $H$ -color  $G$  ?*

The complexity of the  $H$ -coloring problem for graphs  $H$  has been widely studied [10]. It is shown in [9] that the  $H$ -coloring problem is polynomial time solvable if  $H$  is bipartite or contains a loop, and is NP-complete otherwise. A number of variants of this basic family of problems have been considered, and in [8] the authors have set up a framework for these variants. One of the parameters in this framework is a restriction to graphs with a given upper bound on the vertex degrees. For graphs with degrees bounded by 3, some  $H$ -coloring problems that are NP-complete in general, become polynomial time solvable. However, these problems tend to be NP-complete again when the degree bound is 4. Such is, for instance, the situation with  $H = K_3$ , i.e. with 3-colorings. The theorem of Brooks ensures that a connected graph with degrees at most 3, other than  $K_4$ , is 3-colorable [3]. Thus 3-colorability of such graphs is decidable in polynomial time. However, it is known that the 3-colorability of graphs with degrees at most 4 is NP-complete; this follows (via line graphs) from the result of Holyer [11] that deciding whether the chromatic index of a graph is at most 3 is NP-complete. (Our results also imply this fact, see Section 4.) Based on this, and additional evidence of this kind, Feder, Hell and Huang conjectured that any variant of the  $H$ -coloring problem which is NP-complete without degree constraints is also NP-complete with degree constraints, provided the degree bound is high enough [4]. This has been confirmed for list  $H$ -colorings of graphs in [8], and for  $H$ -colorings of graphs in [16].

In contrast, for digraphs  $H$ , the boundary between easy and hard  $H$ -coloring problems is not well-understood. Some partial results have been published in [14,1,7,2] and in several other papers, but it is still not known whether all directed  $H$ -coloring problems are polynomial or NP-complete. This statement is in fact equivalent to the well-known Dichotomy Conjecture of CSP [5]. A refined version of the Dichotomy Conjecture, known as the Dichotomy Classification Conjecture has been proposed in [12]: it claims a concrete classification of  $H$ -coloring problems as polynomial versus NP-complete. It has been shown in

[13,15] that the Dichotomy Classification Conjecture implies the conjecture of Feder, Hell and Huang for  $H$ -coloring of digraphs. We focus on obtaining the precise degree bounds which ensure the NP-completeness, in the case of the three smallest digraphs  $H$  with NP-complete  $H$ -coloring problems. In particular, our results underscore the fact that both the bounds on in- and out-degrees must be sufficiently large.

Note that  $K_3$  can be viewed as a symmetric digraph, namely the digraph  $C$  in Figure 1, and in this context  $C$ -coloring of a digraph  $G$  is precisely a 3-coloring of the underlying undirected graph of  $G$ . The digraphs  $A, B$ , and  $C$ , in Figure 1, are the smallest three digraphs  $H$  with NP-complete  $H$ -coloring problems. All other digraphs  $H$  with three vertices have polynomial-time solvable  $H$ -coloring problems [1,14]. (All digraphs  $H$  with fewer than three vertices have  $H$ -coloring problems that can be solved in polynomial time by 2-SAT [10].) We investigate these first three interesting cases. We confirm that the  $H$ -coloring problems for these three digraphs,  $H = A, B$ , and  $C$ , are polynomial time solvable for digraphs in which at least one of in-degree or out-degree is 1, and become NP-complete again when both in- and out-degree bounds are at least 2.

Specifically, let  $\Delta^+$  denote the maximum out-degree and  $\Delta^-$  the maximum in-degree in a digraph. Consider the class of digraphs with  $\Delta^+ \leq a$ ,  $\Delta^- \leq b$ . We show that if  $\min(a, b) = 1$ , then there are polynomial time  $A$ -coloring,  $B$ -coloring, and  $C$ -coloring algorithms, but if  $\min(a, b) \geq 2$ , then all three problems are again NP-complete for the class. The NP-completeness of  $C$ -coloring will imply the theorem of Holyer [11] mentioned earlier.

For the polynomial cases, we provide algorithms for the more general problem of *list  $H$ -coloring*.

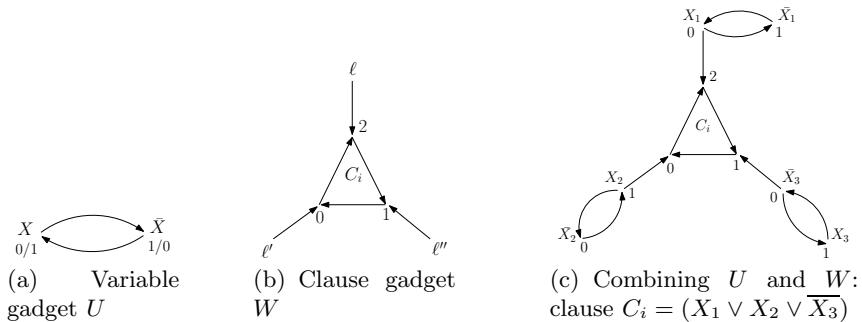
## 2 General Digraphs

In this section, we will present new NP-completeness proofs for the  $A$ -coloring,  $B$ -coloring, and  $C$ -coloring of general digraphs, i.e. without any degree restrictions. It is a folklore result that these three problems are NP-complete [1,14]. The proofs we provide here will facilitate extensions to graphs with bounded degrees.

As is standard in such proofs, we will have a *variable gadget* for each variable and a *clause gadget* for each clause. We will reduce the problem 1-in-3-SAT to the  $A$ - and  $B$ -coloring problems, i.e., for each conjunctive normal formula  $\phi$  with three literals per clause, we produce a graph  $G_\phi$  in such a way that  $\phi$  has a truth assignment with exactly one true literal in each clause if and only if  $G_\phi$  has an  $H$ -coloring. For the  $C$ -coloring problem, we will show a simialr reduction from the problem 3-SAT.

We begin with  $A$ -coloring. We construct a reduction from 1-in-3-SAT as follows. Let  $\phi$  be a given 3-CNF formula. Consider the variable gadget  $U$  shown in Figure 2(a). For each variable  $X$  in  $\phi$ , we have one such gadget, with the two endpoints of  $U$  corresponding to the variable  $X$  and its negation  $\overline{X}$ .

Figure 2(b) depicts the clause gadget  $W$ , for a generic clause  $C_i = (\ell \vee \ell' \vee \ell'')$  with three literals  $\ell, \ell', \ell''$ . The vertices  $\ell, \ell', \ell''$  are identified with the same

**Fig. 2.** NP-completeness of  $A$ -coloring

literals of the corresponding variable gadgets, producing a graph  $G_\phi$ . Note that an endpoint of a variable gadget has an outgoing edge for each occurrence of the corresponding variable (or its negation) in a clause of  $\phi$ .

The graph  $A$  has only one digon, with the vertices 0 and 1. Thus, there are exactly two ways of  $A$ -coloring the variable gadget  $U$ , with  $X$  colored either 0 or 1 and  $\bar{X}$  colored 1 or 0, respectively, as depicted in Figure 2(a). The inner 3-cycle of the clause gadget  $W$  requires three different colors, and in fact the colors must appear in the cyclic order 0, 2, 1, as depicted in the figure, up to the symmetry of  $W$ . Consider now the possible colors of the vertices  $\ell, \ell', \ell''$  (up to symmetry):

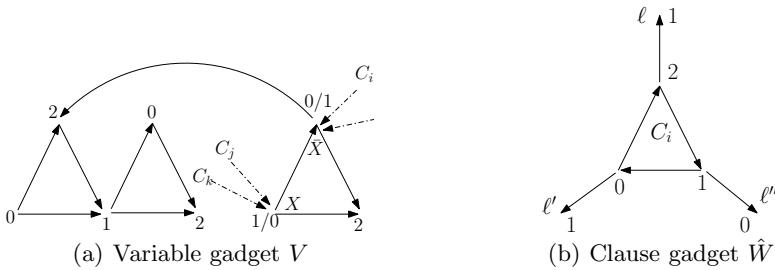
- (0,0,0) : as 00 is not an edge, this would not allow a 0 on the inner cycle;
- (1,1,1) : as 11 is not an edge, this would not allow a 1 on the inner cycle;
- (0,1,1) : as 10 is the only edge from 1, this would force two 0's on the inner cycle;
- (0,0,1) : Figure 2(c) depicts the unique  $A$ -coloring

It follows that in any  $A$ -coloring of  $G_\phi$ , exactly one literal in each clause is colored by 1. Thus if we assign the value True to the literals colored 1 and value False to the literals colored 0, we obtain a satisfying truth assignment for  $\phi$ . Conversely, if we start with a truth assignment and color all vertices of the variable gadgets 0 if the literal is False and 1 if the literal is True, there is, in each clause, exactly one literal that is colored 1, and so the above analysis shows that the colors can be extended to the inner cycles of all the clause gadgets. In conclusion,  $\phi$  is satisfiable if and only if  $G_\phi$  is  $A$ -colorable.

Since the  $A$ -coloring problem is clearly in NP, we have the following theorem.

**Theorem 1.** *The  $A$ -coloring problem is NP-complete.*

A similar analysis applies for  $B$ -coloring. The variable gadget  $V$  is shown in Figure 3(a). It is more complex than for  $A$ -coloring, but there are still two vertices corresponding to the variable  $X$  and its negation  $\bar{X}$ . It is important to observe that  $V$  can be  $B$ -colored so that  $X, \bar{X}$  are colored by 0,1 or by 1,0, but no other pair of colors. To see this, note that the vertex lying in the two



**Fig. 3.** NP-completeness of  $B$ -coloring

adjoining triangles must be colored 1 in any  $B$ -coloring of  $V$ , thus forcing the colors of the adjoining triangles of  $V$  as depicted in the figure. Therefore, the color of  $\overline{X}$  is either 0 or 1 (since it has an out-neighbor colored 2). On the other hand, the color of  $X$  cannot be 2, since  $X$  has two adjacent out-neighbors. Thus coloring  $\overline{X}$  by 0 forces  $X$  to be colored 1 and vice versa, and no other pair of colors for  $X, \overline{X}$  is possible.

Figure 3(b) depicts the clause gadget  $\hat{W}$  for  $C_i = (\ell \vee \ell' \vee \ell'')$  of the 3-CNF formula. This gadget is similar to the gadget  $W$  used for  $A$  coloring, except the edges now point away from the center cycle, into the variable gadgets. In any  $B$ -coloring, the inner cycle requires three different colors. Again, colors in the order of 0, 2, 1, as depicted, give the only  $B$ -coloring for such a cycle (up to symmetry). The vertices  $\ell, \ell', \ell''$  are identified with the corresponding literals. They can only be colored 0 or 1. This again gives rise to four possibilities for the colors of  $\ell, \ell', \ell''$  (up to symmetry), but here only (0,1,1) is a valid choice:

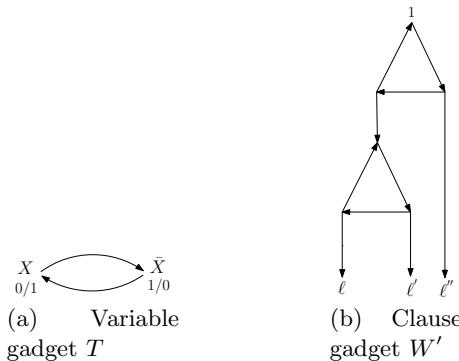
1. (0,0,0) : as 00 is not an edge, this would not allow a 0 on the inner cycle;
2. (1,1,1) : as 11 is not an edge, this would not allow a 1 on the inner cycle;
3. (0,0,1) : as 20 is not an edge, this would force two 1's on the inner cycle;
4. (0,1,1) : Figure 3(b) depicts the unique  $B$ -coloring.

Now, if we assign a truth value of True to the literals that are colored 0, and False to the literals that are colored 1, we have a satisfying assignment for the 1-in-3-SAT problem, and vice-versa. Thus, we have the following theorem.

**Theorem 2.** *The  $B$ -coloring problem is NP-complete.*

Finally, we handle the case of  $C$ -coloring.

Recall that a  $C$ -coloring of a digraph  $G$  is precisely a 3-coloring of the underlying undirected graph of  $G$ . Thus we can use the proof from [6], which constructs a reduction from 3-SAT using the clause gadget depicted in Figure 4(b). The variable gadget depicted in Figure 4(a), and the identification of the literals  $\ell, \ell', \ell''$  with the same literals of the appropriate clause gadgets of  $G_\phi$  are made as before. Assuming the top vertex of the clause gadget is colored 1 as depicted, the three vertices labelled  $\ell, \ell', \ell''$  (corresponding to the literals) cannot be colored by 0,0,0, but can be colored by any other combination of 0,1. Now it is easy

**Fig. 4.** NP-completeness of  $C$ -coloring

to see that an instance  $\phi$  of 3-SAT is satisfiable if and only if the corresponding digraph  $G_\phi$  (i.e., its underlying undirected graph) is 3-colorable cf. [6].

**Theorem 3.** *The  $C$ -coloring problem is NP-complete.*

### 3 Bounded Degree Digraphs

We observed in the previous section that the  $A$ -,  $B$ -, and  $C$ -coloring problems are NP-complete for general digraphs. Now, we investigate the same problems restricted to the class of digraphs with bounded in- and out-degrees.

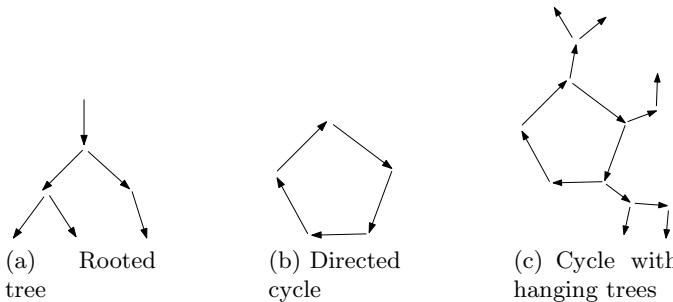
Let  $H$  be any fixed digraph. We begin by presenting a polynomial-time  $H$ -coloring algorithm for digraphs  $G$  with degree bound  $\Delta^- \leq 1$ , or degree bound  $\Delta^+ \leq 1$ . This will imply that there are polynomial-time algorithms for any restriction to digraphs with degree bounds  $\Delta^+ \leq a$ ,  $\Delta^- \leq 1$ , or  $\Delta^+ \leq 1$ ,  $\Delta^- \leq b$ , where  $a, b$  are any positive integers.

A connected digraph  $G$  with degree bound  $\Delta^- \leq 1$ , or degree bound  $\Delta^+ \leq 1$  is either a rooted tree or a directed cycle with rooted trees attached at vertices of the cycle (see Figure 5(c) illustrating the case  $\Delta^+ \leq 2$ ,  $\Delta^- \leq 1$ ). Oriented cycles and multiple directed cycles in a single component are not possible since they would violate the degree requirement at some vertex.

As mentioned earlier, we actually solve a more general problem called list  $H$ -coloring. To begin with, let us introduce it formally. Given a fixed digraph  $H$ , the problem of *list  $H$ -coloring* is the following:

*Given an input graph  $G$ , and for each  $v \in V(G)$ , a list  $L(v) \subseteq V(H)$ , is there a homomorphism  $f : G \rightarrow H$  such that  $f(v) \in L(v)$  for all vertices  $v \in V(G)$ ?*

We note that by setting all lists  $L(v) = V(H)$ , we reduce the  $H$ -coloring problem to the list  $H$ -coloring problem. Thus, solving the list  $H$ -coloring problem in polynomial time also solves the  $H$ -coloring problem in polynomial time.



**Fig. 5.** Bounded digraphs:  $\Delta^+ \leq 2$ ,  $\Delta^- \leq 1$

*Algorithm.* *Arc Consistency* is a basic technique from artificial intelligence, useful for solving list homomorphism problems [10]. In particular, it consists of considering an edge  $uv$  of the input graph  $G$  and reducing the lists  $L(u), L(v)$  so that for each  $x \in L(u)$  there exists a  $y \in L(v)$  with  $xy \in E(H)$ , and for each  $z \in L(v)$  there exists a  $w \in L(u)$  with  $wz \in E(H)$ . This kind of constraint propagation will make the entire graph arc consistent by repeating this removal as long as possible, while the lists in  $G$  are changing. Since  $H$  is fixed, after linearly many updates we obtain final lists satisfying all these constraints. It is easy to observe that if there is at least one empty final list, there is no homomorphism from  $G$  to  $H$  satisfying the original lists.

When  $G$  is a tree and the final lists are all non-empty, there is a list homomorphism  $G \rightarrow H$  satisfying the original lists - simply choose one element from the final list of the root vertex and propagate this choice to all other vertices using the fact that both constraints were satisfied. This is not true, however, in general. For instance, when  $H$  is the 2-cycle with edges 01, 10 and  $G$  is a directed cycle of odd length (see Figure 5(b)) with all lists equal to  $\{0, 1\}$ , then the conditions are satisfied but there is no homomorphism. If we choose one vertex of the 5-cycle and map it to 0, say, then the constraints will propagate around the cycle and arrive requiring the choice 1 at the chosen vertex (and vice versa). However, if at least one vertex of a cycle has a list of size one, the constraint propagation around the cycle will work properly.

With this knowledge in mind, we propose our list  $H$ -coloring algorithm:

1. Run an arc-consistency algorithm over all the edges of  $G$ .
2. If some vertex has an empty list after the arc-consistency process, then, there is no list homomorphism.
3. Else, find an  $H$ -coloring for each weak component of  $G$  separately.
  - If the component is a tree, choose one vertex and map it to one member of its list and propagate this choice to the entire component.
  - If the component is a directed cycle, choose one vertex  $u$ , and consider  $|L(u)|$  subproblems, in which the list  $L(u)$  is reduced to a single vertex of  $H$ , i.e. for each of the possible  $L(u)$  choices. Perform constraint propagation

through the component based on this choice. If at least one choice leads to a homomorphism, then we have a solution, otherwise there is no solution.

- If the component has a directed cycle with hanging trees, follow the procedure to find a homomorphism for the directed cycle first. Then, extend it to find a homomorphism for each of the hanging trees.

**Lemma 1.** *Let  $H$  be a fixed digraph, and let  $a, b$  be any positive integers. The list  $H$ -coloring problem is polynomial time solvable for digraphs with  $\Delta^+ \leq a$ ,  $\Delta^- \leq 1$ , or  $\Delta^+ \leq 1$ ,  $\Delta^- \leq b$ .*

In particular, this applies to  $H = A, B, C$ .

We now focus on input digraphs with  $\Delta^+ \leq a$ ,  $\Delta^- \leq b$ , where both  $a, b$  are at least 2. In order to prove the NP-completeness of this bounded degree setting, we only need to prove the conclusion for  $a = b = 2$ . We shall make modifications to the gadgets used in the unbounded degree case, as the variable gadgets used in Section 2 could potentially have very high outdegree or indegree at the literal vertices. So, to keep the degree constraints satisfied, we need to construct multiple copies of each literal to join the clause gadgets of various clauses in which the literal occurs. At the same time, we have to ensure that the color assigned to the literal vertex remains same in all copies, so that we have a consistent truth value assignment.

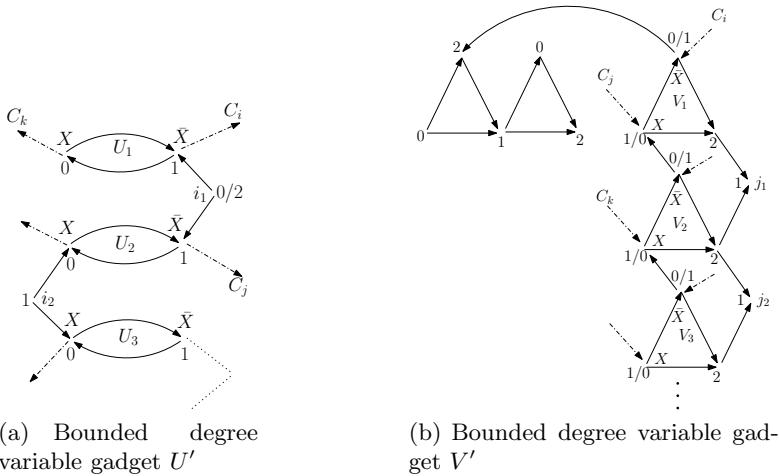
**Lemma 2.** *The  $A$ -coloring problem is NP-complete, even when restricted to digraphs with  $\Delta^+ \leq 2$ ,  $\Delta^- \leq 2$ .*

Consider the variable gadget  $U$  used in Figure 2(a). If a literal occurs  $x$  times in  $\phi$ , then the NP-completeness construction gives the corresponding vertex in  $U$  an out-degree of  $x + 1$ . Therefore, we shall consider a modified variable gadget  $U'$ , see Figure 6(a), which has multiple copies of each literal vertex. If  $X$  occurs  $x$  times in  $\phi$ , and  $\overline{X}$  occurs  $x'$  times in  $\phi$ , then we construct  $U'$  using digons  $U_1, U_2, \dots, U_k$ , where  $k = \max(x, x')$ .

As before, in any  $A$ -coloring each digon must be colored by 0 and 1. Because of the vertex  $i_1$ , the first two copies of  $\overline{X}$  must be colored by the same color, 0 or 1. (Indeed, there is no vertex in  $A$  that has an edge to both 0 and 1.) Similarly, the vertex  $i_2$  ensures that the second and third copy of  $X$  are colored by the same color. Repeating this argument we conclude that all vertices corresponding to  $X$  have the same color, and similarly for  $\overline{X}$ . This allows us to apply the previous proof to a new graph  $G_\phi$  in which each literal vertex in  $U'$  is adjacent to at most one clause gadget  $W$  (the same clause gadget as before). Thus  $G_\phi$  has all in- and out-degrees bounded by 2. It follows from the above remarks that each  $A$ -coloring of  $G_\phi$  yields a satisfying truth assignment of  $\phi$ , and it is also easy to see that any satisfying truth assignment of  $\phi$  can be extended to an  $A$ -coloring of  $G_\phi$ .

The analysis is again similar for  $B$ -coloring.

**Lemma 3.** *The  $B$ -coloring problem is NP-complete, even when restricted to digraphs with  $\Delta^+ \leq 2$ ,  $\Delta^- \leq 2$ .*

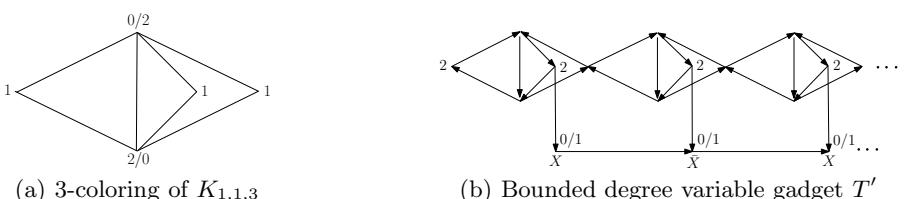


**Fig. 6.**  $A$ - and  $B$ -coloring for  $\Delta^+ \leq 2, \Delta^- \leq 2 \in \text{NP-complete}$

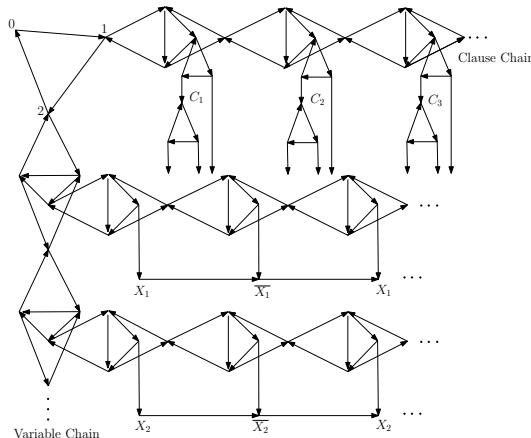
In this case, we consider construct  $G_\phi$  from the modified variable gadget  $V'$  in Figure 6(b), and the same clause gadget  $\hat{W}$  as before. If  $X$  occurs  $x$  times in  $\phi$ , and  $\bar{X}$  occurs  $x'$  times in  $\phi$ , then we construct  $V'$  using triangles  $V_1, V_2, \dots, V_k$ , where  $k = \max(x, x')$ , allowing for a sufficient number of copies of  $X$  and  $\bar{X}$  to keep the in- and out-degrees bounded by 2. It remains to verify that the repeated copies of  $X$  and  $\bar{X}$  must obtain the same color in any  $B$ -coloring of  $G_\phi$ . As noted earlier, the third vertex of  $V_1$  must be colored 2, and hence the vertex  $j_1$  must be colored 1. This means that the third vertex of  $V_2$  is colored 0 or 2; but 0 is not possible as the vertex has in-degree two and 20 is not an edge of  $B$ . Thus the third vertex of each  $V_i$  is 2. It now follows that in any  $B$ -coloring of  $V'$  all copies of  $X$  (and all copies of  $\bar{X}$ ) receive the same color, 0 or 1. Therefore, we can repeat the previous proof of NP-completeness.

Finally, we discuss  $C$ -coloring.

**Lemma 4.** *The  $C$ -coloring problem is NP-complete, even when restricted to digraphs with  $\Delta^+ \leq 2, \Delta^- \leq 2$ .*



**Fig. 7.**  $C$ -coloring for  $\Delta^+ \leq 2, \Delta^- \leq 2 \in \text{NP-complete}$



**Fig. 8.** The overall NP-completeness construction of  $G_\phi$

Here, we use the variable gadget depicted in Figure 7(b). It is an orientation of a chain of copies of  $K_{1,1,3}$ , and  $K_{1,1,3}$  is a graph that takes the same color for every vertex of degree two. Thus assuming the leftmost vertex is colored 2 as depicted, all the vertices corresponding to  $X$  and  $\bar{X}$  are colored by 0 and 1 or 1 and 0 respectively. The overall construction is depicted in Figure 8. There is a basis triangle from which the construction emanates, and which is assumed to be colored by 0, 1, 2 as depicted (by renaming the colors if necessary). There is a *clause chain* (of copies of  $K_{1,1,3}$ ) providing us with a sufficient number of vertices colored 1 for each clause  $C_i$  of  $\phi$ . We use the same clause gadget  $W'$  as in the unbounded setting. Finally, there is a *variable chain* (of copies of  $K_{1,1,3}$ ) yielding a sufficient number of vertices colored 2 for each variable  $X_j$ . The literal vertices of the clause gadgets are identified with the corresponding literals in the variable chains as before. It is easy to check that the maximum in- and out-degree is two, and that  $\phi$  is satisfiable if and only if  $G_\phi$  is 3-colorable.

Combining all the results for the restricted setting of bounded degree input graphs, we have our main theorem.

**Theorem 4.** *Deciding A-, B-, and C-colorability for input digraphs with  $\Delta^+ \leq a, \Delta^- \leq b$  is in P if  $\min(a, b) = 1$ , and is NP-complete if  $\min(a, b) \geq 2$ .*

## 4 Conclusions

There are three smallest digraphs  $H$  with NP-complete  $H$ -coloring problems, namely  $A, B$ , and  $C$  from Figure 1. We have shown that for all three, the  $H$ -coloring problem remains NP-complete if the in- and out-degrees are bounded from above by any constants larger than or equal to 2, but become polynomial-time solvable if at least one of these degree bounds is lowered to 1. This underscores

the fact that the degree bounds must be high enough for both the in-degree and the out-degree: no degree bound on in-degrees ensures NP-completeness if the out-degree bound is 1, and vice versa.

We conclude by noting that our proof implies the following classical result of Holyer. The proof is exactly the same as for Lemma 4, except the edges of the gadgets are not oriented.

**Theorem 5.** [11] *The 3-coloring problem is NP-complete even when restricted to graphs with maximum degree 4.*

**Acknowledgements.** The authors thank the MITACS Globalink Internship Program for undergraduate students for funding the internship of the second author which made this collaboration possible. The authors would also like to acknowledge the hospitality of Simon Fraser University, and especially of the IRMACS center, where most of this was research was done, during the research internship of the second author. Finally, the authors are very grateful to the referees for excellent suggestions that improved the focus of this paper.

## References

1. Bang-Jensen, J., Hell, P., MacGillivray, G.: The complexity of colouring by semi-complete digraphs. *SIAM J. Discret. Math.* 1(3), 281–298 (1988)
2. Barto, L., Kozik, M., Niven, T.: Graphs, polymorphisms and the complexity of homomorphism problems. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, STOC 2008, pp. 789–796. ACM, New York (2008)
3. Brooks, R.L.: On coloring the nodes of a network. In: Mathematical Proceedings of the Cambridge Philosophical Society, pp. 194–197 (1941)
4. Feder, T., Hell, P., Huang, J.: List homomorphisms of graphs with bounded degrees. *Discrete Mathematics* 307(3-5), 386–392 (2007)
5. Feder, T., Vardi, M.Y.: The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.* 28(1), 57–104 (1999)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1990)
7. Hell, P., Zhu, X.: Duality and polynomial testing of tree homomorphisms. *Trans. Amer. Math. Soc.* 348, 1281–1297 (1996)
8. Hell, P., Huang, J.: Counting list homomorphisms and graphs with bounded degrees. *Discrete Math.* (2001)
9. Hell, P., Nešetřil, J.: On the complexity of  $H$ -coloring. *J. Comb. Theory Ser. B* 48(1), 92–110 (1990)
10. Hell, P., Nešetřil, J.: Graphs and Homomorphisms. Oxford University Press (2004)
11. Holyer, I.: The NP-completeness of edge-coloring. *SIAM J. Comput.* 10(4), 718–720 (1981)
12. Jeavons, P.: On the algebraic structure of combinatorial problems. *Theoretical Computer Science* 200, 185–204 (1998)

13. Jonsson, P., Krokhin, A., Kuivinen, F.: Hard constraint satisfaction problems have hard gaps at location 1. *Theoretical Computer Science* 410(38-40), 3856–3874 (2009)
14. Maurer, H.A., Sudborough, J.H., Welzl, E.: On the complexity of the general coloring problem. *Information and Control* 51(2), 128–145 (1981)
15. Nešetřil, J., Siggers, M.H., Zádori, L.: A combinatorial constraint satisfaction problem dichotomy classification conjecture. *Eur. J. Comb.* 31(1), 280–296 (2010)
16. Siggers, M.H.: Dichotomy for bounded degree  $H$ -coloring. *Discrete Appl. Math.* 157(2), 201–210 (2009)

# Bounded Model Checking for Propositional Projection Temporal Logic <sup>★</sup>

Zhenhua Duan<sup>1</sup>, Cong Tian<sup>1,2,★</sup> Mengfei Yang<sup>2</sup>, and Jia He<sup>1</sup>

<sup>1</sup> ICTT and ISN Lab, Xidian University, Xi'an, 710071, P.R. China

<sup>2</sup> China Academy of Space Technology, Beijing, 100094, P.R. China

**Abstract.** This paper presents a bounded model checking approach for propositional projection temporal logic (PPTL). To this end, first PPTL is briefly introduced. Then, bounded semantics of PPTL is defined according to its semantics in logic theory. Further, a reduction method from BMC to SAT is given in detail. In addition, an example is presented to illustrate how the approach works. Our experience shows that BMC approach for PPTL proposed in the paper is useful and feasible.

## 1 Introduction

Model checking was firstly proposed by Clarke and Emerson [7] as well as Quielle and Sifakis [11] independently. As a trusted, strong and automatic verification technique, model checking has been widely used in many fields such as verification of hardware, software and communication protocols. Model checking is a procedure used to check whether a property specified by a temporal logic formula  $\phi$  is satisfied by a given system  $M$  defined in a transition structure ( $M \models \phi$ ?). With temporal logics, linear-time temporal logic (LTL)[2] and branching-time temporal logic (CTL)[7] are popular in practice. In particular, they are usually used as description logics to specify properties of systems in model checking. SPIN [9] based on LTL and SMV [13] depended on CTL are two well-known model checkers. However, for a given system, the number of states of the system increases exponentially with the increasing of the number of components in the system, leading to state explosion problem. To combat this problem, many optimization methods are put forward. Symbolic model checking [12,13] aims at reduction in memory needed and high efficiency in model checking. Compositional model checking [17] optimizes model checking by dividing the model under consideration into several parts and conducts the verification of each part respectively to reduce the complexity of the checking process. Bounded model checking (BMC) is an important progress in formalized verification after symbolic model checking [3]. It proves to be an efficient and successful approach. The main idea of BMC is to check whether a property is satisfied by a system with limitation that the searching length is bounded by a given integer  $k$ . If the property is not satisfied, an error is found. Otherwise, we cannot tell whether the

---

\* This research is supported by the NSFC Grant No. 61003078, 61133001, and 61272117, 973 Program Grant No. 2010CB328102 and ISN Lab Grant No. ISN1102001.

\*\* Corresponding author.

system satisfies the property or not. In this case, we can consider increasing  $k$ , and perform the process of BMC again. In the procedure, BMC is reduced into a SAT problem to find a solution. Although SAT is proved to be an NP-complete problem, however, SAT solver works well in many practical applications with the DPLL algorithm[14]. BMC applies temporal logics as its underlying logics to specify properties of systems as the basic model checking does. Theories and supporting tool NuSMV2 [1] based on LTL is now available. Other tools that can conduct BMC are Bounded Model Checker of CMU [10], Thunder of Intel [8] and so on.

With model checking and bounded model checking, the mostly used temporal logics are LTL, CTL and their variations. An LTL formula is assigned along the path. If all paths from a given state  $s$  satisfies an LTL formula  $f$ , we say  $s$  satisfies  $f$ . Thus, LTL implicitly limits all the paths with a universal quantifier. Therefore, assertions that whether or not there exists a path satisfied the property cannot be specified with LTL. Branching-time temporal logic solved this problem by allowing the use of path quantifier explicitly. From this point, we could say CTL is more expressive than LTL. However, CTL cannot choose a range of paths by means of a formula, which indicates LTL is more expressive than CTL. In addition, there exist two limitations in the grammar of CTL. Boolean combination of path formulas is not allowed in CTL as well as the nestification of path modal operators  $X$ ,  $F$  and  $G$ . CTL\* is a logic which is of the expressiveness of LTL and CTL. Since LTL lacks the path quantifiers while CTL lacks the ability of LTL to specify a single path elaborately [15], CTL\* is more expressive than both of the logics. However, the cost of computation for CTL\* is quite high.

Since the expressiveness of LTL and CTL is not powerful enough, actually, not full regular, therefore, there are at least two types of properties in practice which cannot be specified by LTL and CTL: (1) some time duration related properties such as property  $P$  holds at 100<sup>th</sup> state or  $P$  holds after 100<sup>th</sup> time unites and before 200<sup>th</sup> time unites; (2) some repeated properties (kleen closure) $P$ . Propositional projection temporal logic (PPTL)[21] is an extension of interval temporal logic (ITL)[16]. The expressiveness of PPTL is full regular[18] which allows us to verify full regular properties and time duration related properties of systems in a convenient way. For instance, the above mentioned time duration properties can be defined as  $\text{len}100; P$  and  $\text{len}100; (\diamond P \wedge \text{len}100)$  respectively while the closure property can be defined by  $P^*$ .

With the community of projection temporal logic (PTL), plenty of logic laws [24] have been formalized and proved, and a decision procedure for checking satisfiability of PPTL formulas has also been given [22,23]. Further, a model checker based on SPIN for PPTL has been developed [5]. Therefore, full regular properties and time duration properties of systems can be automatically verified with SPIN. However, since the state explosion problem is a common problem for model checking, the model checker based on SPIN for PPTL also suffers from this problem. To combat it, we are motivated to investigate bounded model checking for PPTL in this paper. To do so, the bounded semantics of PPTL is defined, and some lemmas and theorems for building relationship between bounded model checking and the basic model checking are proved. Based on these basic theories, BMC for PPTL is reduced into SAT problem. To illustrate how our approach can be used to verify full regular properties of systems, an example for verifying feasibility of rate monotonic scheduling (RMS) algorithm [4] is presented

in details. To realize our bounded model checking approach, a prototype of bounded model checker based on NuSMV2 has also been developed recently.

The rest of the paper is organized as follows. PPTL is briefly introduced in section 2. In section 3, bounded model checking for PPTL is formalized in detail. Further, an example for verifying feasibility of RM scheduling algorithm is demonstrated. Some related work are reviewed in section 4. Finally, the conclusion is drawn in section 5.

## 2 Propositional Projection Temporal Logic

To study bounded model checking of interval based temporal logic (IBTL), we use proposition projection temporal logic (PPTL) as the underlying logic. The syntax and semantics of PPTL is briefly introduced in the following. The details of the logic can refer to [22].

### 2.1 Syntax

Let  $Prop$  be a countable set of atomic propositions. PPTL formulas can be inductively defined as below:

- 1) Every atomic proposition  $p \in Prop$  is a formula;
- 2) If  $P, P_1, \dots, P_m$  are PPTL formulas, so are the following constructs:  $\neg P$ ,  $P_1 \vee P_2$ ,  $\bigcirc P$ , and  $(P_1, \dots, P_m) \text{ prj } P$ .

A formula without temporal operators ( $\bigcirc, \text{prj}$ ) is called a state formula, otherwise it is a temporal formula.

### 2.2 Semantics

**States:** Following the definition of Kripke's structure, we define a state  $s$  over  $Prop$  as a mapping from  $Prop$  to  $B = \{\text{true}, \text{false}\}$ ,  $s : Prop \rightarrow B$ . We use  $s[p]$  to represent the valuation of  $p$  at state  $s$ .

**Intervals:** An interval  $\sigma$  is a non-empty sequence of states, which can be finite or infinite. The length of  $\sigma$ ,  $|\sigma|$ , is  $\omega$  if  $\sigma$  is infinite, and the number of states minus 1 if  $\sigma$  is finite. To have a uniform notation for both finite and infinite intervals, we will use extended integers as indices. That is, we consider the set  $N_0$  of non-negative integers and  $\omega$ ,  $N_\omega = N_0 \cup \{\omega\}$ , and extend the comparison operators,  $=, <, \leq$ , to  $N_\omega$  by considering  $\omega = \omega$ , and for all  $i \in N_0, i < \omega$ . Moreover, we define  $\leq$  as  $\leq -(\omega, \omega)$ . To simplify definitions, we will denote  $\sigma$  by  $< s_0, \dots, s_{|\sigma|} >$ , where  $s_{|\sigma|}$  is undefined if  $\sigma$  is infinite. With such a notation,  $\sigma_{(i..j)} (0 \leq i \leq j \leq |\sigma|)$  denotes the sub-interval  $< s_i, \dots, s_j >$  and  $\sigma^{(k)} (0 \leq k \leq |\sigma|)$  denotes  $< s_k, \dots, s_{|\sigma|} >$ . The concatenation of a finite  $\sigma$  with another interval (or empty string)  $\sigma'$  is denoted by  $\sigma \cdot \sigma'$ .

Let  $\sigma = < s_0, \dots, s_{|\sigma|} >$  be an interval and  $r_1, \dots, r_h$  be integers ( $h \geq 1$ ) such that  $0 \leq r_1 \leq r_2 \leq \dots \leq r_h \leq |\sigma|$ . The projection of  $\sigma$  onto  $r_1, \dots, r_h$  is the interval (namely projected interval)  $\sigma \downarrow (r_1, \dots, r_h) = < s_{t_1}, s_{t_2}, \dots, s_{t_l} >$ , where  $t_1, \dots, t_l$  is obtained from  $r_1, \dots, r_h$  by deleting all duplicates. That is,  $t_1, \dots, t_l$  is the longest strictly increasing subsequence of  $r_1, \dots, r_h$ . For instance,  $< s_0, s_1, s_2, s_3, s_4 > \downarrow (0, 0, 2, 2, 2, 3) = < s_0, s_2, s_3 >$ .

**Interpretations:** An interpretation is a triple  $\mathcal{I} = (\sigma, i, j)$ , where  $\sigma$  is an interval,  $i$  is an integer, and  $j$  an integer or  $\omega$  such that  $i \leq j \leq |\sigma|$ . We use the notation  $(\sigma, i, j) \models P$  to denote that formula  $P$  is interpreted and satisfied over the subinterval  $< s_i, \dots, s_j >$  of  $\sigma$  with the current state being  $s_i$ . The satisfaction relation ( $\models$ ) is inductively defined as follows:

- $I - prop$   $\mathcal{I} \models p$  iff  $s_i[p] = true$ , and  $p \in Prop$  is an proposition
- $I - not$   $\mathcal{I} \models \neg P$  iff  $\mathcal{I} \not\models P$
- $I - or$   $\mathcal{I} \models P \vee Q$  iff  $\mathcal{I} \models P$  or  $\mathcal{I} \models Q$
- $I - next$   $\mathcal{I} \models \bigcirc P$  iff  $i < j$  and  $(\sigma, i+1, j) \models P$
- $I - prj$   $\mathcal{I} \models (P_1, \dots, P_m) prj P$ , if there exist integers  $r_0 \leq r_1 \leq \dots \leq r_m \leq j$  such that  
 $(\sigma, r_0, r_1) \models P_1$ ,  $(\sigma, r_{l-1}, r_l) \models P_l$ ,  $1 < l \leq m$ , and  $(\sigma', 0, |\sigma'|) \models Q$  for one  
of the following  $\sigma'$  :  
(a)  $r_m < j$  and  $\sigma' = \sigma \downarrow (r_0, \dots, r_m) \cdot \sigma_{(r_m+1..j)}$ , or  
(b)  $r_m = j$  and  $\sigma' = \sigma \downarrow (r_0, \dots, r_h)$  for some  $0 \leq h \leq m$ .

**Satisfaction and Validity:** A formula  $P$  is satisfied by an interval  $\sigma$ , denoted by  $\sigma \models P$ , if  $(\sigma, 0, |\sigma|) \models P$ . A formula  $P$  is called satisfiable if  $\sigma \models P$  for some  $\sigma$ . A formula  $P$  is valid, denoted by  $\models P$ , if  $\sigma \models P$  for all  $\sigma$ .

### 2.3 Normal Form

Normal Form (NF) [22] is a useful notation in our methods. We assume  $Q_p$  is the set of atomic propositions appearing in the PPTL formula  $Q$ . Then, the NF can be defined as follows:

**Definition 1.** *Normal Form of  $Q$ :  $NF(Q) \equiv \bigvee_{j=1}^{n_0} (Q_{ej} \wedge empty) \vee \bigvee_{i=1}^n (Q_{ci} \wedge \bigcirc Q'_i)$ .*

To simplify the proof and expressiveness, we sometimes use  $Q_e \wedge empty$  instead of  $\bigvee_{j=1}^{n_0} (Q_{ej} \wedge empty)$  and apply  $\bigvee_{i=1}^r (Q_i \wedge \bigcirc Q'_i)$  to replace  $\bigvee_{i=1}^n (Q_{ci} \wedge \bigcirc Q'_i)$ . Thus,  $NF(Q) \equiv Q_e \wedge empty \vee \bigvee_{i=1}^r (Q_i \wedge \bigcirc Q'_i)$ , where  $Q_e$  and  $Q_i$  are state formulas.

**Theorem 1.** *Any PPTL formula  $P$  can be rewritten into its Normal Form.*

The proof of this theorem can be found in [22].

## 3 Bounded Model Checking for PPTL

In this section, we investigate the bounded model checking of PPTL based on the theory presented in section 2. To do this, we first establish the basic theory, bounded semantics, of bounded model checking of PPTL. Then we introduce a new approach of bounded model checking for PPTL. Since each PPTL formula can be transformed into an equivalent formula in NF, unlike LTL and CTL formulas [3,19], we do not need to deal with all types of PPTL formulas in the bounded semantics. As a result, we need only consider formulas with  $\vee, \neg$ , and  $\bigcirc$  operators as well as *empty* and *more*. In the following, we define the bounded semantics of PPTL formulas. Let  $\sigma$  be an interval and  $P$  a PPTL formula.

### 3.1 Bounded Semantics

In this subsection, we first define Kripke structure, then we define bounded semantics of PPTL formulas.

**Definition 2.** A Kripke structure  $M$  is a quadruple  $M = (S, I, T, L)$  where  $S$  is the set of states,  $I \subseteq S$  is the set of initial states,  $T \subseteq S \times S$  is the transition relation and  $L : S \rightarrow P(A)$  is the labeling function, where  $A$  is the set of atomic propositions and  $P(A)$  denotes the powerset over  $A$ .

We use Kripke structure to model systems, and PPTL formulas to define properties of systems. A path in a Kripke structure can be treated as an interval. Since a path with a Kripke structure can be finite or infinite so an interval can also be finite or infinite. Therefore, in the following, we give the definition of  $k$ -loop in an interval.

**Definition 3. ( $k$ -loop)** For  $l, k \in N_0$  and  $l \leq k$ , if there is a translation from  $s_k$  to  $s_l$  in  $\sigma$  and  $\sigma = (s_0, \dots, s_{l-1}) \cdot (s_l, \dots, s_k)^\omega$ , we call interval  $\sigma$  a  $(k, l)$ -loop. If there exist  $k, l$ ,  $k \geq l \geq 0$  such that  $\sigma$  is a  $(k, l)$ -loop, we call  $\sigma$  a  $k$ -loop.

Obviously, if  $\sigma$  is an infinite interval with loop, it must be a  $k$ -loop for some  $k \in N_0$ . An interval with  $k$ -loop is shown in Fig.1. Now we can define the successor of state  $s_i$  over an interval  $\sigma$  by  $\text{succ}(i)$  below:

$$\text{succ}(i) = \begin{cases} i+1 & \text{if } i < k \\ l & \text{if } i = k \text{ and } \sigma \text{ is an infinite interval with } (k, l)-\text{loop} \\ k+1 & \text{if } i = k \text{ and } \sigma \text{ is a finite interval} \end{cases}$$

To define bounded semantics of PPTL formulas, given a bound  $k \in N_0$ , we define an interpretation to be a pair  $I' = (\sigma, i)$ , where  $\sigma$  is an interval,  $i$  an integer and  $0 \leq i \leq k \leq |\sigma|$ . We use the notation  $(\sigma, i) \models_k P$  to represent that formula  $P$  is interpreted and satisfied over either the whole interval if  $\sigma$  is a  $k$ -loop, or subinterval  $< s_i, \dots, s_k >$  of  $\sigma$  otherwise, with the current state being  $s_i$ . In terms of the semantics of PPTL formulas given in section 2, in general,  $(\sigma, i) \models_k P$  is not equal to  $(\sigma, i, k) \models P$ .

The bounded satisfaction relation  $\models_k$  is inductively defined as follows:

$$\begin{aligned} (\sigma, i) \models_k p &\quad \text{iff } s_i[p] = \text{true if } p \in \text{Prop is an atomic proposition} \\ (\sigma, i) \models_k \neg P &\quad \text{iff } (\sigma, i) \not\models_k P \\ (\sigma, i) \models_k P_1 \vee P_2 &\quad \text{iff } (\sigma, i) \models_k P_1 \text{ or } (\sigma, i) \models_k P_2 \\ (\sigma, i) \models_k \bigcirc P &\quad \text{iff } \text{succ}(i) \leq k \text{ and } (\sigma, \text{succ}(i)) \models_k P \\ (\sigma, i) \models_k \text{empty} &\quad \text{iff } i = k \\ (\sigma, i) \models_k \text{more} &\quad \text{iff } i < k \end{aligned}$$

A formula  $P$  is satisfied with bound  $k$  by an interval  $\sigma$ , denoted by  $\sigma \models_k P$ , if  $(\sigma, 0) \models_k P$ . It is not difficult to see the following fact:

**Fact 1.** For a PPTL formula  $P$ , if  $\sigma$  is finite,  $(\sigma, i) \models_k P \iff (\sigma, i, k) \models P$

As any PPTL formula  $P$  can be transformed into its normal form, the above definition about satisfaction relation in bounded semantics of PPTL is sound.

### 3.2 Bounded Model Checking

According to the structure of interval  $\sigma$ , two basic lemmas are given below:

**Lemma 1.** Let  $k \in N_0$ ,  $f$  be a PPTL formula and  $\sigma$  a  $k$ -loop. Then  $\sigma \models_k f$  iff  $\sigma \models f$ .

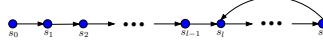


Fig. 1.  $\sigma$  is a  $k$ -loop

Note that we do not need to deal with *empty* and *more* in the above inductive cases since *empty*  $\equiv \neg \bigcirc \text{true}$  and *more*  $\equiv \bigcirc \text{true}$ .

**Lemma 2.** Let  $f$  be a PPTL formula, and  $\sigma$  a finite interval. Then  $\sigma \models f \Rightarrow \exists k, k \geq 0, \sigma \models_k f$ .

**Theorem 2.** Suppose a system is described by a Kripke structure  $\mathbf{M}$ , and some property is defined by a PPTL formula  $f$ . let  $M$  be the set of all intervals generated in  $\mathbf{M}$ . We define  $M \models \exists f$  iff  $\exists \sigma \in M$  and  $\sigma \models f$  as well as  $M \models_k \exists f$  iff  $\exists \sigma \in M$  and  $\sigma \models_k f$ . Then we have  $M \models \exists f$  iff  $\exists k, k \geq 0, M \models_k \exists f$ .

**Proof:** We need prove that  $\exists \sigma \in M$  and  $\sigma \models f$  iff  $\exists \sigma \in M$  and  $\sigma \models_k f$ . To do so, we consider intervals in  $M$  as infinite or finite ones respectively.

1. Suppose  $\sigma$  is infinite, there must be an integer  $k \geq 0$  that makes  $\sigma$  a  $k$ -loop. By Lemma 1:  $\sigma \models_k f$  iff  $\sigma \models f$ . We have,  $M \models \exists f$  iff  $\exists k, k \geq 0, M \models_k \exists f$ .
2. Suppose interval  $\sigma$  is finite, i.e.,  $|\sigma| = m \in N_0$ .

$$\begin{aligned} " \Rightarrow " : M \models \exists f &\Leftrightarrow \exists \sigma \in M \wedge \sigma \models f \\ &\Leftrightarrow \exists k, k \geq 0, \exists \sigma \in M \wedge \sigma \models_k f \\ &\Leftrightarrow \exists k, k \geq 0, \Sigma \models_k \exists f \quad (\text{Lemma 2}) \\ " \Leftarrow " : \exists k, k \geq 0, M \models_k \exists f &\Leftrightarrow \exists k, k \geq 0, \exists \sigma \in M \wedge \sigma \models_k f \\ &\Leftrightarrow \exists k, k \geq 0, \exists \sigma \in M \wedge (\sigma, 0) \models_k f \\ &\Leftrightarrow \exists k, k \geq 0, \exists \sigma \in M \wedge (\sigma, 0, k) \models f \quad (\text{Fact 1}) \end{aligned}$$

Let  $\sigma' = \langle s_0, s_1, \dots, s_k \rangle$ , we have  $(\sigma', 0, k) \models f$ . Since  $\sigma'$  is a subinterval of  $\sigma$  and  $|\sigma'| = k$ , hence,  $\exists \sigma' \in M$  and  $\sigma' \models f$ , leading to  $M \models \exists f$ . As a result,  $\exists k, k \geq 0, M \models_k \exists f \Rightarrow M \models \exists f$ .  $\square$

### 3.3 Reducing BMC of PPTL to SAT

Given a finite state transition system  $M$  (a Kripke structure or NFG), the property of the system in terms of a PPTL formula  $f$ , and a bound  $k$ , then the procedure of BMC can be described as a process for constructing a proposition formula  $[M, f]_k$ . Let  $(s_0, \dots, s_k)$  be a subinterval of interval  $\sigma$ . The definition of formula  $[M, f]_k$  consists

of two components:  $M_k$ , the first component, is a propositional formula that constrains  $(s_0, \dots, s_k)$  to be a valid interval starting from an initial state;  $X_k$ , the second component, a propositional formula that constrains  $\sigma$  to satisfy  $f$  with bound  $k$ . To define the second component, we first give the definition of *loop condition*, which is a propositional formula that is evaluated to true only if the interval  $\sigma$  contains a loop; then we give some translation rules for formulas.

**Definition 4.** For a Kripke structure  $M$ , and  $k \geq 0$ ,  $M_k \stackrel{\text{def}}{=} I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$ .

Loop condition:  $L_{(k,l)} \stackrel{\text{def}}{=} T(s_k, s_l)$  ( $0 \leq l \leq k$ ) and  $L_k \stackrel{\text{def}}{=} \bigvee_{l=0}^k L_{(k,l)}$ .

Let  $p$  be an atomic proposition, and  $f, g$  be PPTL formulas,  $\sigma$  an interval with a  $(k, l)$ -loop,  $s_i$  the current state, and  $k$  the bound, where  $0 \leq i, l, k \leq |\sigma|$ ,  $i, l \leq k$ . The translation of a PPTL formula is given below:

**k-Loop:** Translation  $f(i, k, l)$  of a PPTL formula  $f$  over an infinite interval  $\sigma$  with  $(k, l)$ -loop is inductively defined as follows.

$$p_{(i,k,l)} \stackrel{\text{def}}{=} p(s_i) \text{ if } p \in \text{Prop} \text{ is an atomic proposition, meaning } s_i[p]$$

$$\begin{aligned} (\neg f)_{(i,k,l)} &\stackrel{\text{def}}{=} \neg f_{(i,k,l)} & (f \vee g)_{(i,k,l)} &\stackrel{\text{def}}{=} f_{(i,k,l)} \vee g_{(i,k,l)} & (\bigcirc f)_{(i,k,l)} &\stackrel{\text{def}}{=} f_{(\text{succ}(i),k,l)} \\ \text{empty}_{(i,k,l)} &\stackrel{\text{def}}{=} \begin{cases} \text{false} & i < k \\ \text{true} & i = k \end{cases} & \text{more}_{(i,k,l)} &\stackrel{\text{def}}{=} \begin{cases} \text{true} & i < k \\ \text{false} & i = k \end{cases} \end{aligned}$$

For the translation presented above, a new propositional variable is introduced for each intermediate formula  $e_{(i,k,l)}$ , where  $e$  is a sub-formula of PPTL formula  $f$  and  $0 \leq i \leq k$  and  $l$  indicates the existence of  $(k, l)$ -loop. If  $e$  is an atomic proposition or its negation,  $e_{(i,k,l)}$  is substituted by a boolean variable with the truth-value at state  $s_i$ . Otherwise,  $e_{(i,k,l)}$  indicates a propositional formula whose satisfiability should be considered over the interval  $(s_i, \dots, s_k)$ .

**No Loop:** Translation  $f(i, k)$  of a PPTL formula  $f$  for a finite interval  $\sigma$  (with no loop).

$$\begin{aligned} p_{(i,k)} &\stackrel{\text{def}}{=} p(s_i) & (\neg f)_{(i,k)} &\stackrel{\text{def}}{=} \neg f_{(i,k)} & (f \vee g)_{(i,k)} &\stackrel{\text{def}}{=} f_{(i,k)} \vee g_{(i,k)} \\ (\bigcirc f)_{(i,k)} &\stackrel{\text{def}}{=} f_{(i+1,k)} & f_{(k+1,k)} &\stackrel{\text{def}}{=} \text{false} & & \\ \text{empty}_{(i,k)} &\stackrel{\text{def}}{=} \begin{cases} \text{false} & i < k \\ \text{true} & i = k \end{cases} & \text{more}_{(i,k)} &\stackrel{\text{def}}{=} \begin{cases} \text{true} & i < k \\ \text{false} & i = k \end{cases} \end{aligned}$$

When the intermediate formula  $e_{(i,k)}$  appears in a PPTL formula for a finite interval (with no loop), the explanations are similar to the ones for a PPTL formula with an interval with  $(k, l)$ -loop and omitted here.

**General translation (BMC to SAT):**

$$X_k \stackrel{\text{def}}{=} (\neg L_k \wedge f_{(0,k)}) \vee \bigvee_{l=0}^k (L_{(k,l)} \wedge f_{(0,k,l)}) \text{ and } [M, f]_k \stackrel{\text{def}}{=} M_k \wedge X_k, \text{ i.e.,}$$

$$[M, f]_k \stackrel{\text{def}}{=} I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge [(\neg L_k \wedge f_{(0,k)}) \vee \bigvee_{l=0}^k (L_{(k,l)} \wedge f_{(0,k,l)})]$$

As we can see, the right side of the definition can be divided into two parts: the first part  $I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge (\neg L_k \wedge f_{(0,k)})$  indicates an interval with no loop and the translation without loop is used; the second part  $I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge (\bigvee_{l=0}^k (L_{(k,l)} \wedge f_{(0,k,l)}))$  presents that an interval with a  $k$ -loop and all possible starting points  $l$  of a loop and the translation for a  $k$ -loop is conjoined with the corresponding loop condition  $L_{(k,l)}$ .

**Lemma 3.** Let  $k \in N_0$ ,  $f$  be a PPTL formula and  $\sigma$  an interval. The translation  $f_{(i,k,l)}$  or  $f_{(i,k)}$  is defined as above. Then

- (1)  $\sigma \models_k f$  iff  $\sigma \models_k f_{(i,k,l)}$  if  $\sigma$  is infinite with a  $(k, l)$ -loop or
- (2)  $\sigma \models_k f$  iff  $\sigma \models_k f_{(i,k)}$  if  $\sigma$  is finite.

Now we come to prove an important conclusion formalized in Theorem 3.

**Theorem 3.**  $[M, f]_k$  is satisfiable iff  $M \models_k \exists f$ .

**Proof:** " $\Rightarrow$ " Suppose  $[M, f]_k$  is satisfiable. That is,  $\exists k \geq 0, \sigma \in M, \sigma \models [M, f]_k \Leftrightarrow \sigma \models M_k \wedge [(\neg L_k \wedge f_{(0,k)}) \vee \bigvee_{l=0}^k (L_{(k,l)} \wedge f_{(0,k,l)})]$ .

If  $|\sigma| = \omega$  with  $(k, l)$ -loop and  $l \leq k$ , we have,

$$\begin{aligned} \sigma \models [M, f]_k &\Rightarrow \bigvee_{l=0}^k (L_{(k,l)} \wedge f_{(0,k,l)}) && \text{def of } [M, f]_k \\ &\Rightarrow \sigma \models \exists j \ 0 \leq j \leq k \ f_{(0,k,j)} \\ &\Rightarrow \sigma \models f_{(0,k,j)} \ (0 \leq j \leq k) \\ &\Rightarrow \sigma \models_k f_{(0,k,j)} \ (0 \leq j \leq k) && \text{Lem1} \\ &\Rightarrow \sigma \models_k f && \text{Lem3} \\ &\Rightarrow \exists k, k \geq 0, \sigma \in M \ \sigma \models_k f \\ &\Rightarrow M \models_k \exists f && \text{def} \end{aligned}$$

If  $|\sigma| \in N_0$  with no loop, we have,

$$\begin{aligned} \sigma \models [M, f]_k &\Rightarrow \sigma \models f_{(0,k)} && \text{def of } [M, f]_k \\ &\Rightarrow \exists k, k \geq 0, \sigma \models_k f_{(0,k)} && \text{Lem1} \\ &\Rightarrow \exists k, k \geq 0, \sigma \in M \ \sigma \models_k f && \text{Lem3} \\ &\Rightarrow M \models_k \exists f && \text{def} \end{aligned}$$

" $\Leftarrow$ ": Suppose  $M \models_k \exists f$ . Since  $M \models_k \exists f \Leftrightarrow \exists \sigma \in M, \sigma \models_k f$ , so  $k \geq 0$  is the bound, and there exists subinterval  $< s_0, \dots, s_k >$  of  $\sigma \in M$  such that  $\sigma \models_k M_k \equiv I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$ .

If  $|\sigma| = \omega$  with  $(k, l)$ -loop, there is a  $l, 0 \leq l \leq k$  and  $T(s_k, s_l)$ , so  $\sigma \models_k T(s_k, s_l)$ .

$$\begin{aligned} \sigma \models_k f &\Rightarrow \sigma \models_k f_{(0,k,l)} && \text{Lem3} \\ &\Rightarrow \sigma \models_k M_k \wedge L(k, l) \wedge f_{(0,k,l)} \\ &\Rightarrow \sigma \models_k [M, f]_k && \text{def of } [M, f]_k \end{aligned}$$

If  $|\sigma| = \omega$  without loop,  $\sigma \models_k \neg L_k$ , leading to  $\sigma \models_k M_k \wedge \neg L_k$ . Thus, we have,

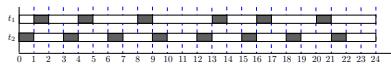
$$\begin{aligned} \sigma \models_k f &\Rightarrow \sigma \models_k f_{(0,k)} && \text{Lem3} \\ &\Rightarrow \sigma \models_k M_k \wedge \neg L_k \wedge f_{(0,k)} \\ &\Rightarrow \sigma \models_k [M, f]_k && \text{def of } [M, f]_k \quad \square \end{aligned}$$

### 3.4 An Example: BMC for RMS

RMS (rate monotonic scheduling) is a classical algorithm for periodic task scheduling proposed by Liu and Layland in 1973. As one of the static priority-driven scheduling

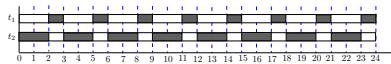
approaches, RMS is optimal. Let  $T = \{t_1, t_2, \dots, t_n\}$  be a set composed of  $n$  periodic tasks. A task is defined as a quadruple  $t_i = (T_i, D_i, E_i, P_i)$ .  $T_i$  is the period of  $t_i$ ,  $D_i$  is the deadline of  $t_i$ ,  $E_i$  denotes the execution time of  $t_i$  and  $P_i$  indicates priority level of  $t_i$ . In general,  $D_i = T_i$  and  $P_i > P_j$  if  $T_i < T_j$ . Thus, the definition can be simplified as a pair  $t_i = (T_i, E_i)$ . Since the utilization factor of CPU is defined as  $U_i = E_i/T_i$ , the utilization factor of CPU for task set  $T$  can be defined as  $U = \sum_{i=1}^n U_i$ . It has been proved by Liu and Layland that for any task set  $T$ , if  $\sum_{i=1}^n U_i \leq n(2^{1/n} - 1)$ ,  $T$  is schedulable with RMS algorithm. However, this is only a unnecessary and sufficient condition.

If a task set  $T$  is schedulable with RMS algorithm, any  $t_i \in T$  must be real-time. This indicates that the remaining execution time of  $t_i$  is 0 or 1 if  $t_i$  is being executed when a new period of  $t_i$  is coming in the next moment, which means the execution time of  $t_i$  in its every period must equals  $E_i$ . To explain this property in detail, we give the following example. Given a task set  $T = \{t_1, t_2\}$ , where  $t_1 = (4, 1)$ ,  $t_2 = (3, 1)$ . Since  $1/4 + 1/3 < 2 \times (\sqrt{2} - 1)$ , so  $T$  is schedulable with RMS algorithm. Since  $T_1 > T_2$ , we have  $P_1 < P_2$ . The execution of  $T$  with RMS algorithm is as shown below in figure 2.



**Fig. 2.** Execution of  $T$  with RMS algorithm

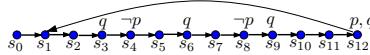
In figure 2, we just simulate 24 time units. From this figure, we can see the task  $T$  is schedulable and the real-time property is satisfied. Alternatively, we alter the task set as  $T_f = \{t_1, t_2\}$  where  $t_1 = (4, 2)$ ,  $t_2 = (3, 2)$ . In this case,  $t_2$  is still prior to  $t_1$ . The execution of  $T_f$  with RMS algorithm is shown in figure 3.



**Fig. 3.** Execution of  $T_f$  with RMS algorithm

Since  $2/4 + 2/3 > 2 \times (\sqrt{2} - 1)$ , we cannot decide whether  $T$  is schedulable with RMS algorithm. As an application of BMC for PPTL, we will check  $T_f$ 's schedulability with RMS algorithm. To do this, first RMS is described by a Kripke structure  $M$ , and the property to be verifies as a PPTL formula  $P$ ; then an integer  $k > 0$  is chosen as the bound; after that, the model checking problem is translated into SAT problem using the approach as described before; finally, a process checking the SAT problem is employed to find a counterexample or to confirm the property is satisfied.

**Modeling of RMS:** To check the schedulability of  $T_f$  using RMS algorithm by means of bounded model checking of PPTL, we first make a model for figure 3 as shown in figure 4, where  $p$  indicates in a period of  $t_1$ , the execution time equals  $E_1$  and  $q$  denotes



**Fig. 4.** Kripke structure of  $T_f$  with RMS algorithm:  $M$

in a period of  $t_2$ , the execution time equals  $E_2$ . In Kripke structure  $M$  shown in Figure 4,  $S = \{s_0, s_1, \dots, s_{12}\}$ ,  $I = \{s_0\}$  and  $T = \{(s_0, s_1), (s_1, s_2), (s_2, s_3), (s_3, s_4), (s_4, s_5), (s_5, s_6), (s_6, s_7), (s_7, s_8), (s_8, s_9), (s_9, s_{10}), (s_{10}, s_{11}), (s_{11}, s_{12}), (s_{12}, s_1)\}$ . The atomic proposition set is  $A = \{p, q, x, y\}$ , and labeling function can be defined as:  $L(s_3) = \{q\}$ ,  $L(s_6) = \{q\}$ ,  $L(s_9) = \{q\}$  and  $L(s_{12}) = \{p, q\}$ . Note that a state without specified propositions implied that the propositions at the state are negative. For instance,  $s_4$  is unspecified, so  $L(s_4) = \emptyset$ , implying  $\{\neg p, \neg q\}$  at state  $s_4$ . To encode the system, we have to add 2 additional atomic propositions  $x$  and  $y$  into the set of atomic proposition, since there are 13 states in  $M$  and  $2^3 < |S| = 13 < 2^4$ . We do not need to specify values for  $x$  and  $y$  in this example since we do not use them. We can set a fixed order of atomic propositions as  $(p, q, x, y)$ , then we can represent every state  $s \in S$  with the boolean vector  $(p, q, x, y)$ .

**Definition of Property:** The property we concern can be defined as follows:

$$p \text{ is true at state } s_{n*T_1} \text{ and } q \text{ is true at state } s_{n*T_2}, 1 \leq n \in N.$$

This is a typical example for showing the limitation of the expressive power of LTL because this property cannot be described by an LTL formula since the property is full regular. Certainly, we can specify this property by a PPTL formula as follows:

$$F \equiv (\bigcirc^{T_1}(p \wedge \varepsilon))^+ \wedge (\bigcirc^{T_2}(q \wedge \varepsilon))^+$$

Let  $T_1 = 4$  and  $T_2 = 3$ , we have,  $F \equiv (\bigcirc^4(p \wedge \varepsilon))^+ \wedge (\bigcirc^3(q \wedge \varepsilon))^+$ . By means of normal form of PPTL formulas, we can work out  $NF(F)$ :  $NF(F) \equiv \bigcirc(\bigcirc^3(p \wedge \varepsilon); (\bigcirc^4(p \wedge \varepsilon))^*) \wedge \bigcirc(\bigcirc^2(q \wedge \varepsilon); (\bigcirc^3(q \wedge \varepsilon))^*)$  and  $NF(\neg F) \equiv \varepsilon \vee \bigcirc\neg(\bigcirc^3(p \wedge \varepsilon); (\bigcirc^4(p \wedge \varepsilon))^*) \vee \bigcirc\neg(\bigcirc^2(q \wedge \varepsilon); (\bigcirc^3(q \wedge \varepsilon))^*)$

**Translation of BMC to SAT:** With BMC for PPTL, we are trying to find a counterexample of the property  $F$ , which means we are looking for a witness for  $\neg F$ . The existence of such a witness indicates that property  $F$  is false. On the other hand, it means the property holds up to the given bound. Assume the bound  $k = 4$ . Unfolding the transition relation, we can get the following formula:

$$M_4 \equiv I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge T(s_2, s_3) \wedge T(s_3, s_4)$$

According to Figure 4, the loop condition  $L_4 \equiv \bigvee_{l=0}^4 L_{(4,l)} \equiv \bigvee_{l=0}^4 T(s_4, s_l)$  is false. We can get the translation for paths with no loops as following:

$$\neg F \equiv [NF(\neg F)]_{(0,4)} \equiv \neg p(s_4) \vee \neg q(s_3)$$

Putting everything together we can get the following boolean formula:

$$\begin{aligned} [M, \neg F]_{(0,4)} &\equiv M_4 \wedge [(\neg L_4 \wedge (\neg F)_{(0,4)}) \vee \bigvee_{l=0}^4 (L_{(4,l)} \wedge (\neg F)_{(0,4,l)})] \\ &\equiv M_4 \wedge [(true \wedge (\neg F)_{(0,4)}) \vee false] \\ &\equiv M_4 \wedge (\neg F)_{(0,4)} \end{aligned}$$

$$\equiv I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge T(s_2, s_3) \wedge T(s_3, s_4) \wedge (\neg p(s_4) \vee \neg q(s_3))$$

**Finding a Witness for the SAT Problem:** As shown in figure 4, the assignment,  $p(s_4) = 0$  satisfies  $[M, \neg F]_{(0,4)}$ . This assignment corresponds to an interval from the initial state to  $s_4$  that violates the property.

With bounded model checking for PPTL, we proved  $T_f$  is non-schedulable with RMS algorithm.

## 4 Related work

Bounded model checking for linear-time and branching-time temporal logics, have been studied in recent years [3,19]. As typical representatives of two kinds of temporal logics, LTL and CTL are often used as the logics to specify properties in the bounded model checking. A useful supporting tool NuSMV2 [1] is based on LTL. Other BMC supporting tools including Bounded Model Checker of CMU [10], Thunder of Intel [8] and so on are also available. Further, researchers also studied the completeness and complexity of bounded model checking for LTL and CTL, which are considered as computational challenges in bounded model checking [6]. Computing completeness threshold ( $\mathcal{CT}$ ) may be an efficient way but it only works for some simple, non-nested properties [3]. Other methods are checking for convergence which only works for properties that can be reduced to an invariant one and fixpoint detection. All the above methods have limitations since the expressiveness of LTL or CTL is not full regular. Compared with linear-time and branching-time temporal logics, however, interval based temporal logic such as ITL or PPTL is more powerful since they are both full regular. Therefore, we use PPTL as our underlying logic to define properties. In addition, a prototype of bounded model checker based on NuSMV2 for PPTL has been developed. This allows us to bounded model checking for PPTL in an automatical way. Our experience shows that the bounded model checking approach presented in this paper for PPTL is feasible and useful.

## 5 Conclusion

We proposed some basic theory of BMC for PPTL including bounded semantics and process of reducing BMC to a SAT problem. We also gave an example to show the process of BMC for PPTL. Although we established the basic theory on bounded model checking for PPTL, however, we have not investigated the completeness and complexity of it. In the future, we will further investigate the completeness and complexity of BMC for PPTL. Moreover, we will also need to improve the prototype of our model checker of BMC for PPTL so that automatical verification can be conducted. In addition, some practical case studies are also indispensable in the future.

## References

1. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 359–364. Springer, Heidelberg (2002)

2. Pnueli, A.: The Temporal Logic of Programs. In: Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, pp. 46–67. IEEE, New York (1977)
3. Biere, A., Clarke, E., et al.: Bounded Model checking. Advances in Computers, vol. 58, pp. 117–148. Academic Press, London (2003)
4. Liu, C.L., Layland, J.W.: Scheduling algorithm for multiprogramming in a hard real-time environment. Journal of the ACM 20(1), 46–61 (1973)
5. Tian, C., Duan, Z.: Model Checking Propositional Projection Temporal Logic based on SPIN. In: Butler, M., Hinchey, M.G., Larrondo-Petrie, M.M. (eds.) ICFEM 2007. LNCS, vol. 4789, pp. 246–265. Springer, Heidelberg (2007)
6. Clarke, E., Kroening, D., et al.: Computational Challenges in Bounded Model Checking. International Journal on Software Tools for Technology Transfer 7(2), 174–183 (2005)
7. Clark, E., Emerson, E.A.: Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. In: Kozen, D. (ed.) Logic of Programs 1981. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)
8. Copty, F., Fix, L., Fraer, R., Giunchiglia, E., Kamhi, G., Tacchella, A., Vardi, M.Y.: Benefits of Bounded Model Checking at an Industrial Setting. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 436–453. Springer, Heidelberg (2001)
9. Holzmann, G.J.: SPIN Model Checker: The Primer and Reference Manual (September 4, 2003)
10. <http://www.cs.cmu.edu/~modelcheck/bmc.html>
11. Quielle, J.P., Sifakis, J.: Specification and verification of concurrent systems in CESAR. In: Dezani-Ciancaglini, M., Montanari, U. (eds.) Programming 1982. LNCS, vol. 137, pp. 337–351. Springer, Heidelberg (1982)
12. Burch, J.R., Clarke, E., McMillan, K.L., Dill, D.L., Hwang, J.: Symbolic Model Checking:  $10^{20}$  States and Beyond. Information and Computation 98(2), 142–170 (1992)
13. McMillian, K.L.: Symbolic Model Checking. Kluwer Academic Publishers (1993) ISBN: 0-7923-9380-5
14. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM 5, 394–397 (1962)
15. Huth, M., Ryan, M.: Logic in Computer Science: Modeling and Reasoning about Systems, 2nd edn., Part 3. China Machine Press (2007) ISBN:978-7-111-21397-0
16. Moszkowski, B.: Reasoning about digital circuits. Ph.D. Thesis. Stanford University, Stanford (1983)
17. Grumberg, O., Long, D.E.: Model checking and modular verification. Journal ACM Transactions on Programming Languages and Systems TOPLAS Homepage Archive 16(3), 843–871 (1994)
18. Tian, C., Duan, Z.: Propositional projection temporal logic, Büchi automata and  $\omega$ -regular expressions. In: Agrawal, M., Du, D.-Z., Duan, Z., Li, A. (eds.) TAMC 2008. LNCS, vol. 4978, pp. 47–58. Springer, Heidelberg (2008)
19. Zhang, W.: Bounded Semantics of CTL and SAT-Based Verification. In: Breitman, K., Cavalcanti, A. (eds.) ICFEM 2009. LNCS, vol. 5885, pp. 286–305. Springer, Heidelberg (2009)
20. <http://www.cs.cmu.edu/~modelcheck/bmc.html>
21. Duan, Z.: An Extended Interval Temporal Logic and a Framing Technique for Temporal Logic Programming. Ph.D. Thesis, University of Newcastle upon Tyne (May 1996)
22. Duan, Z., Tian, C., Zhang, L.: A Decision Procedure for Propositional Projection Temporal Logic with Infinite Models. Acta Informatica 45(1), 43–78 (2008)
23. Duan, Z., Zhang, L.: A Decision Procedure for Propositional Projection Temporal Logic. Technical Report No.1, Institute of Computing Theory and Technology, Xidian University, Xi'an P.R.China (2005)
24. Duan, Z., Yang, X., Kounty, M.: Framed Temporal Logic Programming. Science of Computer Programming 70(1), 31–61 (2008)

# Packing Cubes into a Cube Is NP-Hard in the Strong Sense

Yiping Lu<sup>1</sup>, Danny Z. Chen<sup>2</sup>, and Jianzhong Cha<sup>1</sup>

<sup>1</sup> School of Mechanical / Electronic and Control Engineering, Beijing Jiaotong University,  
Beijing, 100044, China  
[{yplu,jzcha}@bjtu.edu.cn](mailto:{yplu,jzcha}@bjtu.edu.cn)

<sup>2</sup> Department of Computer Science and Engineering, University of Notre Dame,  
Notre Dame, IN 46556, USA  
[dchen@cse.nd.edu](mailto:dchen@cse.nd.edu)

**Abstract.** While the problem of packing two-dimensional squares into a square, in which a set of squares is packed into a big square, has been proved to be NP-complete, the computational complexity of the  $d$ -dimensional ( $d > 2$ ) problems of packing hypercubes into a hypercube remains an open question [5,7]. In this paper, we show that the three-dimensional problem version of packing cubes into a cube is NP-hard in the strong sense.

**Keywords:** Packing Problems, Cube Packing, NP-hardness.

## 1 Introduction

While the problem of packing 2-D squares into a square, in which a set of squares is packed into a big square, has been proved to be NP-complete over 20 years ago [9], to our best knowledge, the computational complexity of the  $d$ -D ( $d > 2$ ) versions of the problem, i.e., packing multiple hypercubes into a hypercube, is still unsettled. The reason is that, unlike in most other packing situations where the lower dimensional problem version constitutes a special case of the higher dimensional problem, there seems to be no obvious way to degenerate a cube packing problem to a square packing problem. (Similarly, there seems to be no obvious way to degenerate the square packing problem to the 1-D knapsack problem, and this is why the NP-completeness of the problem of packing squares into a square needed a separate proof.)

The 2-D square packing problem, the 3-D and  $d$ -D ( $d > 3$ ) hypercube packing problems have been intensely studied (e.g., see [2, 3, 4, 5, 7, 8]). When developing approximation algorithms for cube and hypercube packing problems, Correa and Kenyon [4], and Bansal *et al.* [1] cited the NP-completeness of the 2-D square packing problem [9], but without stating the NP-completeness/NP-hardness status of the cube and hypercube packing problems. Epstein and van Stee [5] guess that NP-hardness also holds for (hyper) cube packing problems, but at the same time they state that this is still an open question. Harren [7] also states it is still an open question.

In this paper, we show that packing multiple cubes into a cube in 3-D is really a NP-hard problem in the strong sense. Our proof is based on a reduction from the 3-partition problem, which is NP-hard in the strong sense [6].

The rest of this paper is organized as follows. In Section 2, we formally state the target problem. In Section 3, we give the NP-hardness proof of the cube packing problem in 3-D. Section 4 briefly concludes the paper.

## 2 The 3-Partition Problem and the Cube Packing Problems

### 2.1 The 3-Partition Problem

The 3-partition problem is known to be NP-hard in the strong sense [6].

#### 3-Partition Problem

INSTANCE: A set of  $3m$  ( $m > 1$ ) positive integers,  $S = \{a_1, a_2, \dots, a_{3m}\}$ , and a bound  $B$ , such that for every  $i$ ,  $1 \leq i \leq 3m$ ,  $B/4 < a_i < B/2$ , and  $\sum_{i=1}^{3m} a_i = mB$ .

QUESTION: Can  $S$  be partitioned into  $m$  disjoint subsets  $S_1, S_2, \dots, S_m$ , such that for each  $j$  with  $1 \leq j \leq m$ , the sum of all elements in  $S_j$  is exactly  $B$ ? Note that the constraints of the problem require that each subset  $S_j$  should have exactly 3 elements of  $S$ .

For 3-partition problem, it can be proved that we can assume that  $m$  is a multiple of 3 without losing the problem's NP-hardness in strong sense (the corresponding proof of this claim is in this paper's full-length version which exceeds length limit but the authors will be glad to provide it upon email requesting). In the rest of this paper, we will assume that  $m$  is a multiple of 3, and write  $m = 3m'$  (where  $m'$  is a positive integer) whenever needed.

### 2.2 The Problem of Packing Cubes into a Cube

The problem of packing cubes into a cube in 3-D is formally stated below.

#### Problem of Packing Cubes into a Cube

INSTANCE: A big cube  $C$  and a set of small cubes,  $L = \{c_1, c_2, \dots, c_n\}$ , whose size parameters are all positive integers.

**QUESTION:** Can all cubes of  $L$  be packed into  $C$  orthogonally without any intersection of interior points between any two cubes of  $L$ ?

### 3 NP-Hardness of the Problem of Packing Cubes into a Cube

**Theorem:** The problem of packing cubes into a cube is NP-hard in the strong sense.

**Proof:** We will construct an instance of the packing-cubes-into-cube problem by a reduction from the 3-partition problem whose parameter  $m$  is a multiple of 3. Since the later problem is NP-hard in the strong sense, if a reduction from the former to the later can be established in polynomial (or even pseudo-polynomial time), then the former will be also NP-hard in the strong sense.

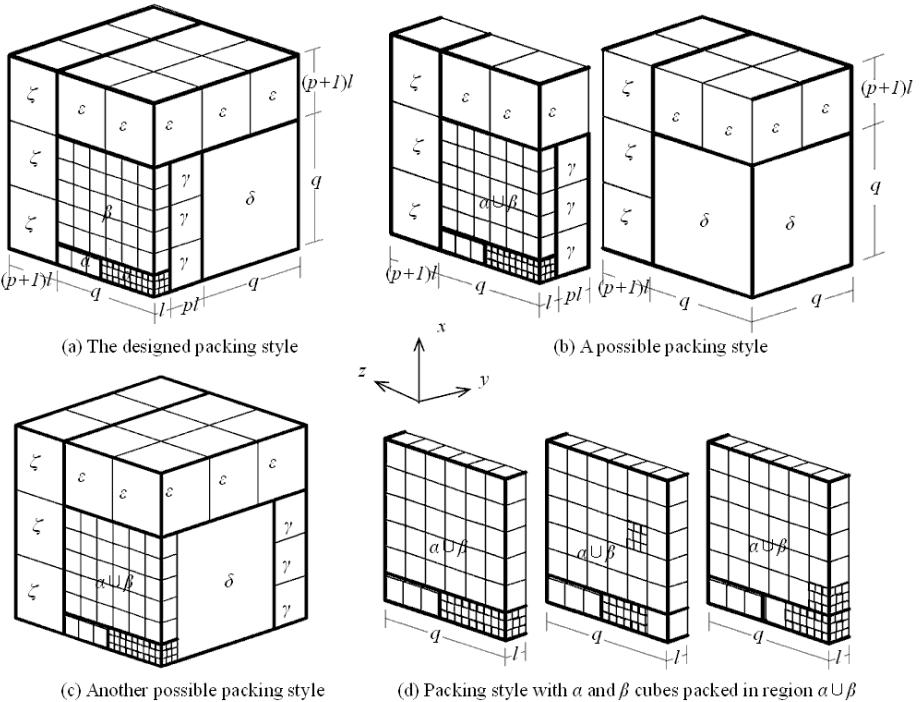
Let  $S = \{a_1, a_2, \dots, a_{3m}\}$  and  $B$  be an instance of the 3-partition problem, where  $m = 3m'$  is a multiple of 3. A corresponding instance of the cube packing problem is defined as follows.

**I. The parameters:** Let  $p, A, l, w, q, h$ , and  $r$  be positive integers that we choose to satisfy:  $p \geq 2$  and  $p(p+1) > m$ ,  $A > p(p+1)B$  and  $A > 6p(p+1)$ ,  $w = 3A + 1.5B$ ,  $l = 3A + B$ ,  $q = p(p+1)l$ ,  $h = m(A+B)$ , and  $r = q - h$ .

**II. The cubic packing space  $C$ :** The size of the cubic packing space  $C$  is  $l + pl + q$ . We actually construct the cubic space  $C$  by using 6 packing regions:  $\alpha, \beta, \gamma, \delta, \varepsilon$ , and  $\zeta$ .

As shown in Fig. 1(a) and Fig. 1(b), the size of region  $\alpha$  is  $l \times w \times q$ . On top of region  $\alpha$ , we put a region  $\beta$  of size  $l \times (q-w) \times q$ . To the right of regions  $\alpha$  and  $\beta$ , we put a region  $\gamma$  of size  $pl \times q \times q$ . To the right of region  $\gamma$ , we put a cubic region  $\delta$  of size  $q \times q \times q$ . On top of regions  $\alpha, \beta, \gamma$ , and  $\delta$ , we put a region  $\varepsilon$  of size  $(l + pl + q) \times (l + pl) \times q$ . To the back of regions  $\alpha, \beta, \gamma, \delta$ , and  $\varepsilon$ , we put a region  $\zeta$  of size  $(l + pl + q) \times (l + pl + q) \times (l + pl)$ .

**III. The list  $L$  of cubic items:** The set  $L$  of packed cubic items consists of the following kinds of cubes (the intuition is to define a set of cubes of different kinds that can be packed into  $C$  by packing the  $\alpha$  cubes into region  $\alpha$ , packing the  $\beta$  cubes into region  $\beta$ , packing the  $\gamma$  cubes into region  $\gamma$ , packing the  $\delta$  cubes into region  $\delta$ , packing the  $\varepsilon$  cubes into region  $\varepsilon$ , and packing the  $\zeta$  cubes into region  $\zeta$ ).

**Fig. 1.** Packing cubes into a cube

❖ The  $\alpha$  cubes: The number of  $\alpha$  cubes is  $N_\alpha = 9m + k$ , including:

- *Type I cubes*:  $3m$  cubes of sizes  $A + a_i$  ( $i = 1, 2, \dots, 3m$ );
- *Type II cubes*:  $2m$  cubes of size  $A + B$  each;
- *Type III cubes*:  $4m$  cubes of size  $A$  each;
- *Type IV cubes*:  $k$  cubes of size  $t_1$  each, where  $k = \lceil r/l \rceil$ ,  $t_1 = r/k$ , and

$$\begin{aligned}
 r &= q - h = p(p+1)l - m(A+B) \\
 &= p(p+1)l - 3m'(A+B) \\
 &= p(p+1)l - m'(3A+B) - 2m'B \\
 &= p(p+1)l - m'l - 2m'B
 \end{aligned}$$

Because  $p(p+1) > m$  and  $A > p(p+1)B$ , we have

$$l = 3A + B > 3mB > 2m'B.$$

Thus

$$\begin{aligned} k &= \lceil r/l \rceil = p(p+1) - m' - \lfloor 2m'B/l \rfloor, \\ &= p(p+1) - m' > m - m/3 = 2m/3, \end{aligned} \quad (1)$$

and

$$\begin{aligned} t_1 &= r/k = l - \frac{2m'B}{p(p+1)-m'} \\ &= 3A + B - \frac{2m'B}{p(p+1)-m'} . \\ &> 3A + B - \frac{2m'B}{3m'-m'} = 3A \end{aligned} \quad (2)$$

Because  $A > 6p(p+1)$ , we have

$$A > 6p(p+1) = 2(m + 3(p(p+1) - m/3)) = 2(m + 3k). \quad (3)$$

- ❖ The  $\beta$  cubes: There are  $N_\beta = p^2(p+1)^2 - p(p+1)$  such cubes of size  $t_2 = l - 0.5B/(p(p+1)-1)$  each. Note that

$$\begin{aligned} t_2 &= l - 0.5B/(p(p+1)-1) \\ &= 3A + B - 0.5B/(p(p+1)-1) > 3A. \end{aligned} \quad (4)$$

- ❖ The  $\gamma$  cubes: There are  $N_\gamma = (p+1)^2$  such cubes of size  $pl$  each.

- ❖ The  $\delta$  cube: There is  $N_\delta = 1$  such cube of size  $q$ .

- ❖ The  $\varepsilon$  and  $\zeta$  cubes: The number of  $\varepsilon$  cubes is  $N_\varepsilon = (p+1)p$  and the number of  $\zeta$  cubes is  $N_\zeta = (p+1)(p+1)$ , whose sizes are  $(p+1)l$  each ( $\varepsilon$  cubes and  $\zeta$  cubes are of the same size).

The above construction can obviously be done in polynomial time, since the total number of cubes thus defined is polynomial in  $m$ . Next, we show that the packing problem instance thus constructed has a “yes” answer if and only if the instance of the 3-partition problem has a “yes” answer.

**The “if” Part:** If the 3-partition problem instance has a “yes” answer, then we show that a packing of the cube  $C$  is achievable. If the answer to the 3-partition problem is “yes”, then we choose to pack the  $\alpha$  cubes into region  $\alpha$ , the  $\beta$  cubes into region  $\beta$ , the  $\gamma$  cubes into region  $\gamma$ , the  $\delta$  cube into region  $\delta$ , the  $\varepsilon$  cubes into region  $\varepsilon$ , and the  $\zeta$  cubes into region  $\zeta$  (as shown in Fig. 1(a)). It can be

proved that packing the  $\alpha$  cubes into region  $\alpha$  is achievable (the proof is in this paper's full-length version which exceeds the length limit but the authors will be glad to provide it upon email requesting). By observing the size definitions of other corresponding cubes and regions, it is easy to check that packing the  $\beta$  cubes into region  $\beta$ , the  $\gamma$  cubes into region  $\gamma$ , the  $\delta$  cube into region  $\delta$ , and the  $\zeta$  cubes into region  $\zeta$  are all achievable.

**The “Only If” Part:** From this point to the end of this proof, we will show that if a packing of the cube  $C$  is achievable, then the answer to the 3-partition problem instance is “yes”.

First, we assume that the achieved packing of  $C$  is in the same style as described above, i.e., the realized packing is in the style of packing all  $\alpha$  cubes into region  $\alpha$ , all  $\beta$  cubes into region  $\beta$ , ..., and all  $\zeta$  cubes into region  $\zeta$  (as shown in Fig. 1(a)). We claim that if this packing style is achievable, then the answer to the 3-partition problem is “yes”. Actually, we can further claim that if the packing style that packs all  $\alpha$ ,  $\beta$ , and  $\gamma$  cubes into the region  $\alpha \cup \beta \cup \gamma$  (i.e., all  $\alpha$ ,  $\beta$ , and  $\gamma$  cubes are packed together into a packing space of size  $(l+pl) \times q \times q$ ), then the answer to the 3-partition problem is “yes”.

Next, we will first prove that if all  $\alpha$ ,  $\beta$ , and  $\gamma$  cubes can be packed into the region  $\alpha \cup \beta \cup \gamma$ , then the answer to the 3-partition problem is “yes”; then, we will show that if packing all the cubic items into the packing space  $C$  is achievable, then all  $\alpha$ ,  $\beta$ , and  $\gamma$  cubes can be packed into the region  $\alpha \cup \beta \cup \gamma$ .

Consider the situation that all  $\alpha$ ,  $\beta$ , and  $\gamma$  cubes are packed into the region  $\alpha \cup \beta \cup \gamma$ . The size of the packing space is  $(p+1)l \times q \times q$ . Because the size of each  $\gamma$  cube is  $pl > (p+1)l/2$ , at most one  $\gamma$  cube can be packed along the  $y$  direction, and it is not difficult to see that if all  $\alpha$ ,  $\beta$ , and  $\gamma$  cubes can be packed into the region  $\alpha \cup \beta \cup \gamma$ , then it must be the case that all  $\alpha$  and  $\beta$  cubes can also be packed into the region  $\alpha \cup \beta$  (this is because in any packing of this particular region  $\alpha \cup \beta \cup \gamma$  using the  $\alpha$ ,  $\beta$ , and  $\gamma$  cubes, all  $\gamma$  cubes can always be shifted to touch one face of the packing space without affecting the packability of the  $\alpha$  and  $\beta$  cubes, see Fig. 1(b)). Now we focus on the packing style that all  $\alpha$  and  $\beta$  cubes are packed into the region  $\alpha \cup \beta$  whose size is  $l \times q \times q$  (as shown in Fig. 1(d)). There are four types of  $\alpha$  cubes: Type I cubes, Type II cubes, Type III cubes, and Type IV cubes — the sizes of the first 3 types of  $\alpha$  cubes are all approximately  $A$ , and the size of the 4<sup>th</sup> type cubes is approximately  $3A$ . Suppose we cut (conceptually) each of the Type IV cubes into 27 equal-size cubes. Then these newly cut cubes all have a size of  $t_1/3$  which is larger than  $A$  (see Equation (2)).

The  $\beta$  cubes all have a size  $t_2$ , which is approximately  $3A$ ; we can also cut

(conceptually) each  $\beta$  cube into 27 equal-size cubes whose size is also larger than  $A$  (see Equation (4)). After these cutting operations, the Types I, II, and III  $\alpha$  cubes, plus the two groups of newly cut cubes, constitute a set of cubes whose sizes are all approximately  $A$ ; we call such cubes the  $A$ -cubes. Clearly, if a packing of all  $\alpha$  and  $\beta$  cubes into the region  $\alpha \cup \beta$  is achievable, then packing all  $A$ -cubes into the same region is also achievable. The minimum size of these  $A$ -cubes is  $A$ , which is equal to the size of the Type III cubes. The total number of all  $A$ -cubes is:

$$\begin{aligned} & 9m + 27k + 27(p^2(p+1)^2 - p(p+1)) \\ &= 27m' + 27(p(p+1) - m') + 27(p^2(p+1)^2 - p(p+1)) \\ &= 27p^2(p+1)^2. \end{aligned}$$

Since  $\frac{q}{A} = \frac{p(p+1)l}{A} = \frac{p(p+1)(3A+B)}{A} = \frac{3p(p+1)A + p(p+1)B}{A}$ ,

and  $A > p(p+1)B$ , we have  $\lfloor q/A \rfloor = 3p(p+1)$ .

Hence, any line parallel to the  $x$  or  $z$  axis can intersect the interior of at most  $3p(p+1)$   $A$ -cubes. Also, because any line parallel to the  $y$  axis can intersect the interior of at most  $\lfloor l/A \rfloor = 3$   $A$ -cubes, the maximum number of  $A$ -cubes that can be packed into the region  $\alpha \cup \beta$  is  $3 \cdot 3p(p+1) \cdot 3p(p+1) = 27p^2(p+1)^2$ , which is exactly the total number of  $A$ -cubes we have. Thus, we know that if the set of  $A$ -cubes can be packed into the cubic space  $l \times q \times q$ , then it must be the case that they can also be packed in a style of a  $3 \cdot 3p(p+1) \cdot 3p(p+1)$  regular grid. Now we focus on the packing space constraint on the  $y$  dimension whose length is  $l$ : If the above packing is achievable, then the grid style packing means that the  $27p^2(p+1)^2$   $A$ -cubes can be partitioned into  $9p^2(p+1)^2$  triplets, each of which contains exactly 3  $A$ -cubes whose size summation is no larger than  $l = 3A + B$ . Since the  $A$ -cubes cut from Type IV  $\alpha$  cubes or from  $\beta$  cubes must stay adjacent to each other in an “actual” packing, we know that the  $9m$  Types I, II, and III  $\alpha$  cubes are partitioned into  $3m$  triplets each of which contains exactly 3  $A$ -cubes whose size summation is no larger than  $l = 3A + B$ . It is not difficult to know that the total size sum of all the  $9m$  Types I, II, and III  $\alpha$  cubes is  $3ml$ , since

$$\sum_{i=1}^{3m} (A + a_i) + 2m(A + B) + 4mA = 9mA + 3mB = 3ml.$$

This means that the sum of cube sizes for each of these  $3m$  triplet must be equal to  $l$  exactly. Since  $l = 3A + B$ , every Type II cube must be in a triplet with two

Type III cubes, and this “forces” the  $3m$  Type I cubes forming  $m$  triplets among themselves.

For every  $j=1, 2, \dots, m$ , suppose the  $j$ -th triplet of Type I cubes is  $A + a_{j1}, A + a_{j2}, A + a_{j3}$ ; then we have

$$A + a_{j1} + A + a_{j2} + A + a_{j3} = l = 3A + B,$$

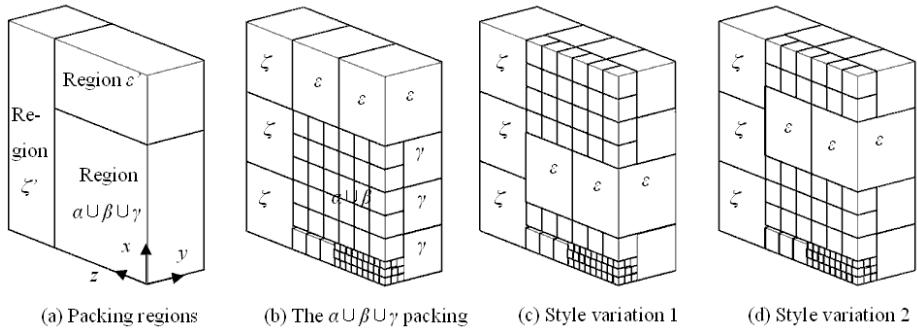
implying  $a_{j1} + a_{j2} + a_{j3} = B$ . Therefore, the answer to the 3-partition problem is “yes”.

Now, we will show that if packing all cubic items into the packing space  $C$  is achievable, then packing all  $\alpha$ ,  $\beta$ , and  $\gamma$  cubes into the region  $\alpha \cup \beta \cup \gamma$  is also achievable. As shown in Fig. 1(a)-(b), it is easy to see that in any achievable packing of  $C$ , the  $\delta$  cube must touch at least two faces of the packing space  $C$ , otherwise there will be some  $\varepsilon$  or  $\zeta$  cubes that cannot be packed (the reason is that the summation of the size of the  $\delta$  cube and the size of an  $\varepsilon$  or  $\zeta$  cube is equal exactly to the size of  $C$ ). Without loss of generality, we assume that the  $\delta$  cube touches the bottom and front faces of  $C$ .<sup>1</sup> Then, if the  $\delta$  cube is not at a corner of  $C$ , then the packing is like the one in Fig. 1(c), where the subregions of  $C$  adjacent to the back face or top face of the  $\delta$  cube are packed with identical  $\varepsilon$  or  $\zeta$  cubes and the subregions of  $C$  adjacent to one of the other two faces of the  $\delta$  cube are packed with  $\gamma$  cubes or  $\alpha$  and  $\beta$  cubes. Further, all  $\alpha$  and  $\beta$  cubes are packed together in a subregion of size  $l \times q \times q$  (as shown in Fig. 1(d)); the reason is that the size of an  $\gamma$  cube is at least 2 times larger than the size of any  $\alpha$  or  $\beta$  cube, and it is easy to see that  $\alpha$  and  $\beta$  cubes cannot be packed on both the left and right faces of the  $\delta$  cube, otherwise there will be at least one  $\gamma$  cube that cannot be packed. If the achieved packing is indeed like the one in Fig. 1(c), then one achievable packing style variation from this style is to have all  $\gamma$  cubes and all  $\alpha$  and  $\beta$  cubes packed on the same side of the  $\delta$  cube. This forms a packing style that has the  $\delta$  cube packed at a corner of  $C$  (as in Fig. 1(a)). When the  $\delta$  cube is packed at a corner of  $C$ , the packing situation must be that of packing all  $\alpha, \beta,$

---

<sup>1</sup> A cubic space or box has 6 faces, as shown in Fig. 1(a). In such a figure, we call its visible faces that are parallel to the  $x \times y$  plane,  $y \times z$  plane, or  $z \times x$  plane the front face, the top face, or the left face, respectively, and call its invisible faces that are parallel to the  $x \times y$  plane,  $y \times z$  plane, or  $z \times x$  plane the back face, the bottom face, or the right face, respectively.

and  $\gamma$  cubes into a subregion  $\alpha \cup \beta \cup \gamma$  of  $C$  (for convenience, we call this packing style an  $\alpha \cup \beta \cup \gamma$  packing, as in Fig. 2(b)). We will show that any other packing style either can be transformed to a feasible  $\alpha \cup \beta \cup \gamma$  packing or cannot exist.



**Fig. 2.** The packing situations of the  $\alpha$ ,  $\beta$ , and  $\gamma$  cubes

So far, we have shown above that the packing style in which the  $\delta$  cube touches only 2 faces of  $C$  can be transformed to an  $\alpha \cup \beta \cup \gamma$  packing. If the  $\delta$  cube is packed at a corner of  $C$ , then the situation is like that in Fig. 1(a)-(b), where all  $\alpha$ ,  $\beta$ , and  $\gamma$  cubes are packed into the region  $\alpha \cup \beta \cup \gamma$ . Since the size of the region  $\alpha \cup \beta \cup \gamma$  along the  $y$  direction is equal to the size of an  $\varepsilon$  or  $\zeta$  cube, without loss of generality, we can assume a situation that on top of the region  $\alpha \cup \beta \cup \gamma$ ,  $\varepsilon$  cubes are packed, and to the back of the region  $\alpha \cup \beta \cup \gamma$ ,  $\zeta$  cubes are packed (as in Fig. 1(b) and Fig. 2). As shown in Fig. 2(a), we call the subregion of  $C$  on top of the region  $\alpha \cup \beta \cup \gamma$  the region  $\varepsilon'$ , and the subregion to the back of the region  $\alpha \cup \beta \cup \gamma$  the region  $\zeta'$ . If the packing style is not an  $\alpha \cup \beta \cup \gamma$  packing, then one possible situation is that some cubes in region  $\varepsilon'$  are packed into the  $\alpha \cup \beta \cup \gamma$  region (as shown in Fig. 2(c)-(d)). In such a situation, the  $\varepsilon$  cubes inside the region  $\alpha \cup \beta \cup \gamma$  will have to be contained by a box space whose size and orientation are equal exactly to those of region  $\varepsilon'$ ; the reason is that the packing of  $\varepsilon$  cubes inside region  $\varepsilon'$  is tight, and thus it requires that the size along the  $z$  direction of the containing box in region  $\alpha \cup \beta \cup \gamma$  filled by such  $\varepsilon$  cubes be a multiple of the size of the  $\gamma$  cubes (otherwise, some  $\gamma$  cubes will not be packable). The smallest of this size is the minimum multiple of the size of  $\varepsilon$  cubes and that of  $\gamma$  cubes, i.e., the minimum multiple of  $(p+1)l$  and  $pl$ , which is  $p(p+1)l = q$  and is equal exactly to the size of region  $\varepsilon'$  along the  $z$  direction. The sizes of this containing box along the  $x$  and  $y$  directions should be equal to the size of the  $\varepsilon$  cubes. This means that if some  $\varepsilon$  cubes in region  $\varepsilon'$  are moved

to and packed in the region  $\alpha \cup \beta \cup \gamma$ , then all the  $\varepsilon$  cubes in region  $\varepsilon'$  must be packed inside the region  $\alpha \cup \beta \cup \gamma$  and actually be packed as if the whole  $\varepsilon'$  region is put into the region  $\alpha \cup \beta \cup \gamma$ . Hence, it is easy to see that any packing situation that has  $\varepsilon$  cubes from region  $\varepsilon'$  packed into the region  $\alpha \cup \beta \cup \gamma$ , like those in Fig. 2(c)-(d), can be transformed to an  $\alpha \cup \beta \cup \gamma$  packing by moving (exchanging) all  $\varepsilon$  cubes inside region  $\alpha \cup \beta \cup \gamma$  to region  $\varepsilon'$ .

Now we consider the situation that on the top and the back of the region  $\alpha \cup \beta \cup \gamma$ , region  $\varepsilon'$  and region  $\zeta'$  are already packed with  $\varepsilon$  and  $\zeta$  cubes, as shown in Fig. 2(b). We will show that if all items inside the region  $\alpha \cup \beta \cup \gamma$  are  $\alpha, \beta$ , and  $\gamma$  cubes, then no  $\varepsilon$  or  $\zeta$  cube from region  $\varepsilon - \varepsilon'$  or  $\zeta - \zeta'$  can be exchanged with and packed into the region  $\alpha \cup \beta \cup \gamma$  (i.e., packing some  $\varepsilon$  or  $\zeta$  cubes into the region  $\alpha \cup \beta \cup \gamma$ , and moving some  $\alpha, \beta$ , or  $\gamma$  cubes to be packed in region  $\varepsilon - \varepsilon'$  or  $\zeta - \zeta'$ ). The reason is that  $\gamma$  cubes are packed tightly inside the region  $\alpha \cup \beta \cup \gamma$  along the  $x$  and  $z$  dimensions; if any block of  $\varepsilon$  or  $\zeta$  cubes is put into the region  $\alpha \cup \beta \cup \gamma$ , then the exchanging packing space must be involved with some  $\gamma$  cubes. This requires that the block sizes (for such  $\varepsilon$  or  $\zeta$  cubes) along both the  $x$  and  $z$  dimensions be a common multiple of the size of  $\gamma$  cubes and the size of  $\varepsilon$  (or  $\zeta$ ) cubes; this makes such sizes of the block be at least the minimum multiple of those two cube sizes, i.e.,  $p(p+1)l = q$ , which is equal exactly to the sizes of the region  $\alpha \cup \beta \cup \gamma$  along the  $x$  and  $z$  dimensions. This means that if any block of  $\varepsilon$  or  $\zeta$  cubes from region  $\varepsilon - \varepsilon'$  or  $\zeta - \zeta'$  is put into the region  $\alpha \cup \beta \cup \gamma$ , then the sizes of such a block will be equal exactly to the sizes of the region  $\alpha \cup \beta \cup \gamma$ , i.e., the whole region of  $\alpha \cup \beta \cup \gamma$  must be exchanged with a whole subregion of  $\varepsilon - \varepsilon'$  or  $\zeta - \zeta'$ . In other words, the  $\alpha, \beta$ , and  $\gamma$  cubes will still be packed into a region whose sizes are equal to those of the region  $\alpha \cup \beta \cup \gamma$ . This may still be considered as an  $\alpha \cup \beta \cup \gamma$  packing. The theorem is thus proved.  $\square$

## 4 Conclusions

We have proved that the problem of packing cubes into a cube in 3-D is NP-hard in the strong sense. This partially settles the open question raised in [5,7]. For the general hypercube packing problem in  $d$ -D ( $d > 3$ ), authors of this paper also foresee a proof, but because of the length limitation, it is not included in this paper.

**Acknowledgement.** The research of D. Z. Chen was supported in part by NSF under Grants CCF-0916606 and CCF-1217906.

## References

1. Bansal, N., Correa, J.R., Kenyon, C., Sviridenko, M.: Bin Packing in Multiple Dimensions: Inapproximability Results and Approximation Schemes. *Mathematics of Operations Research* 31(1), 31–49 (2006)
2. Caprara, A., Lodi, A., Monaci, M.: Fast Approximation Schemes for Two-stage, Two-dimensional Bin Packing. *Mathematics of Operations Research* 30, 136–156 (2005)
3. Chung, F.R.K., Garey, M.R., Johnson, D.S.: On Packing Two-dimensional Bins. *SIAM Journal on Algebraic and Discrete Methods* 3, 66–76 (1982)
4. Correa, J.R., Kenyon, C.: Approximation Schemes for Multidimensional Packing. In: Proc. 15th ACM-SIAM Symposium on Discrete Algorithms, pp. 179–188 (2004)
5. Epstein, L., van Stee, R.: Online Square and Cube Packing. *Acta Informatica* 41(9), 595–606 (2005)
6. Garey, M., Johnson, D.: Computer and Intractability – A Guide to the Theory of NP-Completeness. Freeman, New York (1979)
7. Harren, R.: Approximation Algorithms for Orthogonal Packing Problems for Hypercubes. *Theoretical Computer Science* 410(44), 4504–4532 (2009)
8. Kohayakawa, Y., Miyazawa, F.K., Raghavan, P., Wakabayashi, Y.: Multidimensional Cube Packing. *Algorithmica* 40, 173–187 (2004)
9. Leung, J.Y.-T., Tam, W.T., Wong, C.S., Chin, F.Y.L.: Packing Squares into a Square. *Journal of Parallel and Distributed Computing* 10, 271–275 (1990)
10. Li, K., Cheng, K.H.: Complexity of Resource Allocation and Job Scheduling Problems in Partitionable Mesh Connected Systems. In: Proc. of 1st Annual IEEE Symposium of Parallel and Distributed Processing, Silver Spring, MD, pp. 358–365 (1989)

# On the Complexity of Solving or Approximating Convex Recoloring Problems

Manoel B. Campêlo<sup>1,\*</sup>, Cristiana G. Huiban<sup>2,\*\*</sup>, Rudini M. Sampaio<sup>1</sup>,  
and Yoshiko Wakabayashi<sup>3,\*\*\*</sup>

<sup>1</sup> Universidade Federal do Ceará, Fortaleza, Brazil  
`{mcampelo,rudini}@lia.ufc.br`

<sup>2</sup> Universidade Federal de Pernambuco, Recife, Brazil  
`cmngh@cin.ufpe.br`

<sup>3</sup> Universidade de São Paulo, São Paulo, Brazil  
`yw@ime.usp.br`

**Abstract.** Given a graph with an arbitrary vertex coloring, the Convex Recoloring Problem (CR) consists of recoloring the minimum number of vertices so that each color induces a connected subgraph. We focus on the complexity and inapproximability of this problem on  $k$ -colored graphs, for fixed  $k \geq 2$ . We prove a very strong complexity result showing that CR is already NP-hard on  $k$ -colored grids, and therefore also on planar graphs with maximum degree 4. For each  $k \geq 2$ , we also prove that, for a positive constant  $c$ , there is no  $c \ln n$ -approximation algorithm even for  $k$ -colored  $n$ -vertex bipartite graphs, unless  $P = NP$ . For 2-colored  $(q, q - 4)$ -graphs, a class that includes cographs and  $P_4$ -sparse graphs, we present polynomial-time algorithms for fixed  $q$ . The same complexity results are obtained for a relaxation of CR, where only one fixed color is required to induce a connected subgraph.

**Keywords:** Convex recoloring, NP-hardness, inapproximability, polynomial algorithm, grid graph, cograph,  $P_4$ -sparse graph,  $(q, q - 4)$ -graph.

## 1 Introduction

Consider a game in which all players receive an  $n \times n$  chessboard (grid) in which all the  $n^2$  squares are occupied by either a *black* or a *white* pebble in an arbitrary manner. Every pebble has one face colored black and the other one colored white (as in Reversi or Othello). The initial configuration of the pebbles is the same for all players. The goal of each player is to reverse (turn to the opposite face) a least number of pebbles in order to reach a configuration where each color occupies a unique ‘connected’ region of his board. A winner is a player who makes the least number of reversals.

---

\* Partially supported by FUNCAP and CNPq, Brazil.

\*\* Research conducted while the author was supported by FUNCAP and CNPq.

\*\*\* Partially supported by CNPq (Proc. 303987/2010-3 and 477203/2012-4) and USP MaCLinC/NUMEC Project.

The game we have just described may be seen as the convex recoloring problem on a 2-colored grid. It is an easy to state combinatorial problem, for which it is natural to ask whether an optimal solution can be easily found or not. Surprisingly, it is a hard problem, as we prove in this paper. We shall also prove that the problem remains hard on  $k$ -colored grids, for  $k \geq 3$ .

A  $k$ -coloring of graph  $G$  is a function  $C : V(G) \rightarrow \{1, 2, \dots, k\}$ , and  $C(v)$  is the *color* of  $v$ . A graph is  $k$ -colored if it is assigned a  $k$ -coloring. Note that the coloring considered here differs from the classical (proper) vertex coloring of graphs, where  $C(u) \neq C(v)$  for all  $uv \in E$ . A coloring  $C$  is *convex* if each color class (i.e. set of vertices with the same color) induces a connected subgraph. The *convex recoloring problem* (CR) consists of recoloring (changing the color of) the minimum number of vertices so that the resulting coloring is convex. (We note that, in the recoloring process, some colors may disappear.)

This problem was introduced by Moran and Snir [13,14] in 2005, motivated by studies on phylogenetic trees. They showed that it is NP-hard even on paths, and presented the first approximation results. Since then CR and some of its variants have been intensively investigated. More recently, some applications on routing problems, transportation networks and protein-protein interaction networks have also been mentioned in the literature [7,9]. In general, the idea behind these applications is that, the given colored graph represents some situation, and a convex coloring of such a graph represents a desirable (perfect) configuration, which one wants to achieve after a least number of color changes (such a number is called the *recoloring distance*). In the case of phylogenetic trees, given a set of species, a *perfect phylogenetic tree* on this set is a tree in which the states of each character induce a convex coloring of the tree, where a *character* (coloring) is a biological attribute shared among all the species and a *character state* (color) is the state of this character shown by a vertex [14].

Approximation algorithms for CR were first designed for trees [13,14,4], and more recently generalized to graphs with bounded treewidth [9], for which a  $(2 + \varepsilon)$ -ratio has been shown. When each color appears at most twice, the problem remains NP-hard on paths [10], but admits a  $\frac{3}{2}$ -approximation [12]. For an  $n$ -vertex arbitrary graph, however, CR cannot be approximated within a factor of  $(1 - o(1)) \ln \ln n$  in polynomial time, unless  $\text{NP} \subset \text{DTIME}(n^{O(\log \log n)})$  [9]. On the other hand, some of the approximation algorithms for trees apply to weighted versions of CR and other variants as well (on partial coloring). These more general variants can all be treated (as the cardinality case) using a polyhedral approach [6]. For results in terms of Fixed Parameter Tractability, the reader is referred to [5,16,18].

Here we focus on the problem  $\text{CR}_k$ , the *convex recoloring problem with  $k$  colors in the initial coloring*. Clearly, all approximation algorithms for CR are approximation algorithms for  $\text{CR}_k$ . Only a few hardness results are known for this case. Recently,  $\text{CR}_2$  was proved to be NP-hard on arbitrary graphs [15]. It is very intriguing that only two colors make the convex recoloring already hard. This fact motivated us to investigate nontrivial classes of graphs for which  $\text{CR}_2$  can be solved in polynomial time. While it is easy to see that  $\text{CR}_2$  on trees (resp.

ladders) can be solved in linear (resp. polynomial) time, for the well-structured grid graphs, our intuition that the problem would be easy turned out to be wrong.

The results we present in this paper were motivated by the investigations to understand the polynomial solvability or approximability threshold of the CR2 problem. We show that for any fixed  $k \geq 2$ , CR $k$  is NP-hard on grids, which clearly implies NP-hardness on (bipartite) planar graphs with maximum degree 4. This hardness result motivated us to define a relaxed problem, CR $k$ ONE, a variant of CR $k$  in which only one specified color is required to induce a connected subgraph. This problem, if polynomially solvable, could possibly help in the design of an approximation algorithm for CR $k$ , as it would provide a lower bound for the optimal value of CR $k$ . However, we could prove the same complexity result for this relaxation. In terms of approximability, we show that, for each  $k \geq 2$ , if P  $\neq$  NP, the two problems are not approximable within a factor of  $c \ln n$  on  $n$ -vertex bipartite graphs (for a constant  $c > 0$ ). To our knowledge, this is the strongest inapproximability result for CR $k$  and (and therefore for CR), under the more natural hypothesis that P  $\neq$  NP. Finally, we present algorithms for CR2 and CR2ONE on  $(q, q - 4)$ -graphs, which are polynomial for fixed  $q$ . This class of graphs includes the cographs and  $P_4$ -sparse graphs.

## 2 NP-hardness on Grids

In this section, we first prove that CR $k$  and CR $k$ ONE are NP-hard on grids, for  $k = 2$ . The results are then generalized for any  $k \geq 3$ .

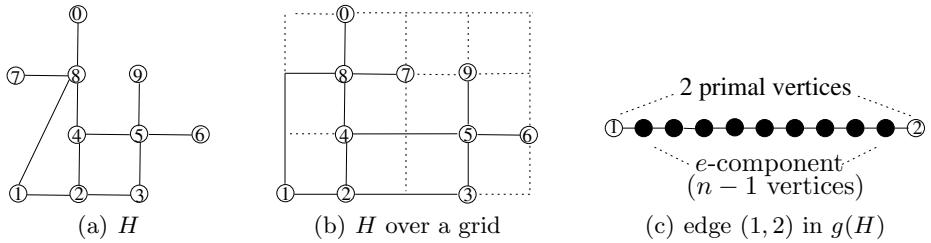
A *grid graph*, or simply a grid, is a graph with vertex set  $V = X \times Y$ , where  $X$  and  $Y$  are finite subsets of consecutive integers, and edge set  $\{(i, j), (i', j')\} : |i - i'| + |j - j'| = 1, i \in X, j \in Y\}$ . If  $m = |X|$  and  $n = |Y|$ , such a grid can also be denoted as  $G(m, n)$ , and considered to have  $m$  rows (or horizontal lines) and  $n$  columns (or vertical lines). We say that a frame is added to  $G(m, n)$  if two extra rows and two extra columns forming a boundary around  $G(m, n)$  are added so as to get a new grid  $G(m + 2, n + 2)$ . Note that a grid is a planar bipartite graph with maximum degree 4.

To obtain the complexity results, we show a reduction from the connected vertex cover problem. A *connected vertex cover* of a graph  $H$  is a subset of vertices that induces a connected subgraph and contains at least one endpoint of each edge of  $H$ . Deciding whether a planar graph with maximum degree at most 4 has a connected vertex cover of size at most a given integer  $p$  is an NP-complete problem [8]. We reduce this problem, denoted as CVC, to the decision version of CR2ONE on grid graphs. Our reduction is inspired by the proof presented in [8].

Let  $H$  be an input graph for CVC,  $n = |V(H)|$  and  $m = |E(H)|$ . Consider a 2-dimensional grid in the oriented plane such that the distance (number of edges) between two consecutive horizontal or vertical lines is  $n$ . Since  $H$  is planar and  $\Delta(H) \leq 4$ , it has a representation where each vertex is placed at a point  $(nx, ny)$ , for some  $x, y \in \mathbb{Z}$ , and the edges are described solely by (horizontal and

vertical) segments of the lines in this grid (see Figure 1). Such a representation can be constructed in low-order-polynomial time [8].

Let  $X$  and  $Y$  be the smallest sequences of integers containing the  $x$ - and  $y$ -coordinates of the vertices of  $H$ , respectively. The instance for CR2ONE is  $g(H) = (G, C, 1)$ , where  $G$  is the grid graph with vertex set  $X \times Y$ ,  $C$  is a 2-coloring (with colors in  $\{1, 2\}$ ), and color 1 is required to induce a connected subgraph. For a vertex  $v$  in  $H$ , we denote by  $\phi(v)$  the corresponding vertex in the grid graph  $G$ , and call it a *primal vertex*. Note that an edge  $e = uv$  of  $H$  corresponds to a path  $P_{uv}$  in  $G$  with at least  $n + 1$  vertices (see Figure 1(c)). The internal section of this path,  $P_{uv} \setminus \{\phi(u), \phi(v)\}$ , is called *e-component*. The coloring  $C$  assigns color 1 to the *e-components* and color 2 to the other vertices.



**Fig. 1.** Reduction from CVC to CR2ONE

**Lemma 1.** [Reduction of CVC to CR2ONE] *Let  $(H, p)$  be an input to CVC, where  $H$  is a connected planar graph with maximum degree at most 4, and  $p$  is a positive integer. Let  $g(H) = (G, C, 1)$  be the corresponding input to CR2ONE. The graph  $H$  has a connected vertex cover with at most  $p$  vertices if, and only if,  $g(H)$  has a solution with at most  $p$  recolored vertices.*

*Proof.* First, assume that  $H$  has a connected vertex cover  $S$  with at most  $p$  vertices. Consider a recoloring of  $G$  in which we change to 1 only the color of each primal vertex  $\phi(s)$ , for  $s \in S$ . This recoloring is clearly a solution to  $g(H)$ , as the vertices with color 1 induce a connected subgraph of  $G$ .

Conversely, suppose that  $g(H)$  has an *optimal* solution of size at most  $p$  that is defined by a set  $R$  of the vertices of  $G$  that had their colors switched. Let  $n = |V(H)|$ . Since  $H$  clearly has a connected cover with  $n - 1$  vertices, we may assume that  $p < n - 1$ . We claim that  $R$  has no vertex from an *e-component*. Indeed, first note that, since each *e-component* has at least  $n - 1$  vertices, the set  $R$  cannot contain all vertices of an *e-component*. Now, if there were an *e-component* containing two adjacent vertices  $u$  and  $v$  such that  $u \in R$  and  $v \notin R$ , recoloring only the vertices in  $R \setminus \{u\}$  would be a better solution. Therefore,  $R$  does not contain any vertex of an *e-component*. Denote by  $G_1$  the connected subgraph of  $G$  induced by the vertices with color 1 in the optimal solution of  $g(H)$  under consideration. We have shown that  $G_1$  contains all the *e-components*. Because

of the minimality of  $R$ , it follows that  $G_1$  is a smallest connected subgraph of  $G$  containing all  $e$ -components. W.l.o.g, we may assume that  $G_1$  has the least number of non-primal vertices. We say that two  $e$ -components  $A$  and  $B$  are neighbors in  $G_1$  if, for all vertices  $u \in A$  and  $v \in B$ , there is a path between them in  $G_1$  that does not include a vertex from other  $e$ -component. So,  $A$  and  $B$  can only be neighbors in  $G_1$  if the corresponding edges in  $H$  are adjacent (otherwise we would need at least  $n - 1$  recolored vertices). Moreover, in this case,  $A$  and  $B$  can always be linked by a single primal vertex. It is not difficult to conclude that  $R$  is the smallest subset of primal vertices connecting the  $e$ -components, and thus, the vertices in  $R$  define in  $H$  a connected vertex cover of size at most  $p$ .  $\square$

The reduction from CVC to CR2 follows an analogous strategy as the reduction from CVC to CR2ONE, but uses a larger initial grid. Owing to space limitation, we only give a sketch of the proof. First, each vertex of  $H$  is placed at a point  $((2n + m)x, (2n + m)y)$ , instead of  $(nx, ny)$ , so that each edge  $e$  is now mapped into a path  $P_e$  with at least  $2n + m + 1$  vertices. The  $e$ -component is now  $P_e$  without *two* vertices in each extremity. So, an  $e$ -component has at least  $2n + m - 3$  (instead of  $n - 1$ ) vertices, and there are 2 (instead of 1) primal vertices associated with each endpoint of  $e$ . Second, a frame is added to the grid (as we defined previously). The instance for CR2 is given by this enlarged grid  $G'$  with a 2-coloring that again assigns color 1 only to the  $e$ -components. By adapting the proof of Lemma 1, we can show that this instance of CR2 has an optimum solution where the subgraph induced by the vertices with color 1 is a tree (this holds because of the duplication of the primal vertices). Then, the boundary (frame) of  $G'$  guarantees the connectedness of the graph induced by the vertices with color 2.

We note that CR $k$  on grids can be reduced to CR( $k + 1$ ). For that, take the  $k$ -colored grid and add a frame colored  $k + 1$ . The same works for CR $k$ ONE. Thus, the following holds:

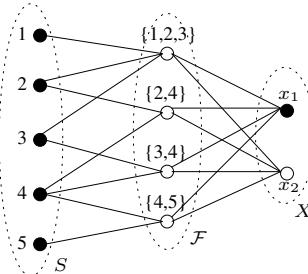
**Theorem 1.** *For every  $k \geq 2$ , CR $k$  and CR $k$ ONE are NP-hard on grids.*

As a consequence of the above theorem, it follows that CR is also NP-hard on grids.

### 3 Approximability Threshold for Bipartite Graphs

We prove in this section an inapproximability result for CR $k$  and CR $k$ ONE on bipartite graphs. We start with  $k = 2$ , showing an approximation preserving reduction from the set cover problem (SC) to CR2. A *cover* of a set  $S$  is a family of subsets whose union comprises  $S$ . Given a set  $S$  and a family  $\mathcal{F}$  of subsets of  $S$ , SC consists of finding a minimum cardinality subfamily of  $\mathcal{F}$  that is a cover of  $S$ . It has been shown that, if P  $\neq$  NP, then SC cannot be approximated in polynomial time within a factor of  $c \ln |S|$ , where  $c$  is a constant [17,1]. This holds even when  $|\mathcal{F}| \leq |S|$ . We use this threshold and the following mapping.

Given an instance  $(S, \mathcal{F})$  of SC, let  $f(S, \mathcal{F})$  be an instance of CR2, consisting of the 2-colored bipartite graph  $(G, C)$ , where  $X = \{x_1, x_2\}$ ,  $V(G) = S \cup \mathcal{F} \cup X$ ,  $E(G) = \{(s, F) \in S \times \mathcal{F} \mid s \in F\} \cup (X \times \mathcal{F})$ ,  $C(v) = 1$ ,  $\forall v \in S \cup \{x_1\}$ , and  $C(u) = 2$ ,  $\forall u \in \mathcal{F} \cup \{x_2\}$ . See Figure 2. We note that this reduction is polynomial in the size of  $(S, \mathcal{F})$ .



**Fig. 2.** Reduction from an instance of SC to an instance of CR2

We can polynomially transform a feasible solution of  $(S, \mathcal{F})$  (given as a subfamily of  $\mathcal{F}$  covering  $S$ ) into a feasible solution of  $f(S, \mathcal{F})$  (described as subset of vertices with switched colors), and vice-versa. In what follows, for an element  $s \in S$ , we denote by  $F_s$  any element of  $\mathcal{F}$  containing  $s$ .

**Lemma 2.** [Reduction from SC to CR2] *Let  $(S, \mathcal{F})$  be an instance of SC and  $f(S, \mathcal{F})$  be the corresponding instance of CR2. If  $\Psi$  is a feasible solution of  $(S, \mathcal{F})$ , then  $\Psi$  is also a feasible solution of  $f(S, \mathcal{F})$ . Conversely, if  $\Psi$  is a feasible solution of  $f(S, \mathcal{F})$ , then  $\Psi'$  is a feasible solution of  $(S, \mathcal{F})$ , where*

$$\Psi' = \begin{cases} \{F_s : s \in S\}, & \text{if } \Psi \cap \mathcal{F} = \emptyset, \\ (\Psi \cap \mathcal{F}) \cup \{F_s : s \in S \cap \Psi\}, & \text{otherwise.} \end{cases}$$

Moreover,  $|\Psi'| \leq |\Psi|$ .

*Proof.* Let  $\Psi$  be a feasible solution of  $(S, \mathcal{F})$ . Since  $\Psi \subseteq \mathcal{F}$  is a cover of  $S$ , there is an edge in  $G$  of the form  $(s, F)$ , for every  $s \in S$  and some  $F \in \Psi$ . Moreover,  $x_1$  is universal to  $\Psi$  in  $G$ . Then, the subgraph of  $G$  induced by the vertices colored 1, that is,  $S \cup \Psi \cup \{x_1\}$ , is connected. The subgraph of  $G$  induced by the vertices colored 2, that is  $\mathcal{F} \setminus \Psi \cup \{x_2\}$ , is also connected because  $x_2$  is universal to  $\mathcal{F}$  in  $G$ .

Now, let  $\Psi$  be a feasible solution of  $f(S, \mathcal{F})$ . If  $\Psi \cap \mathcal{F} = \emptyset$ , then  $\Psi'$  is clearly a cover of  $S$ . In this case, at most one vertex from  $S \cup \{x_1\}$  can retain its original color 1, since they are pairwise non-adjacent in  $G$ . Therefore,  $|\Psi| \geq |S| = |\Psi'|$ . Regarding the case  $\Psi \cap \mathcal{F} \neq \emptyset$ , we have that  $S \cap \Psi$  is trivially covered by the chosen  $F_s$ 's. So, it suffices to show that  $\Psi \cap \mathcal{F}$  covers  $S \setminus \Psi$ . Let  $G_1$  be the connected subgraph of  $G$  induced by the vertices of color 1, after the recoloring defined by  $\Psi$ . Let  $s \in S \setminus \Psi$ . Since  $G_1$  contains  $s$  and the vertices of the nonempty set  $\mathcal{F} \cap \Psi$ , it must also contain an edge  $(s, F) \in E(G)$ , for some  $F \in \mathcal{F} \cap \Psi$  covering  $s$ . Furthermore,  $|\Psi'| \leq |\Psi \cap \mathcal{F}| + |\Psi \cap S| \leq |\Psi \setminus S| + |\Psi \cap S| = |\Psi|$ .  $\square$

**Corollary 1.** *Let  $(S, \mathcal{F})$  be an instance of SC. Optimal solutions of  $(S, \mathcal{F})$  and  $f(S, \mathcal{F}) = (G, C)$  have the same value.*

*Proof.* By Lemma 2, if  $\Psi'$  is a feasible solution of  $(S, \mathcal{F})$ , there is  $\Psi = \Psi'$  that is also a feasible solution of  $f(S, \mathcal{F}) = (G, C)$ . Then, the optimal solution  $|\Psi'^*| = |\Psi| \geq |\Psi^*|$ . So,  $|\Psi'^*| \geq |\Psi^*|$ . The same Lemma also proves that, any solution  $\Psi$  of  $f(S, \mathcal{F}) = (G, C)$  can be mapped into a solution  $\Psi'$  of  $(S, \mathcal{F})$  with  $|\Psi| \geq |\Psi'|$ . Thus,  $|\Psi^*| \geq |\Psi'| \geq |\Psi'^*|$ . So,  $|\Psi^*| \geq |\Psi'^*|$ .  $\square$

For the reduction of SC to CR2ONE, it suffices to consider the graph  $(G - \{x_2\}, C, 1)$ . The counterpart of Lemma 2 readily follows. By this lemma,  $(S, \mathcal{F})$  and  $f(S, \mathcal{F})$  have the same optimal value, and so an  $\alpha$ -approximation for CR2 (or CR2ONE) yields an  $\alpha$ -approximation for SC. The same results hold for  $k \geq 3$ . It suffices to add to  $G$  an isolated vertex  $x_i$  colored  $i$ , for each  $i = 3, 4, \dots, k$ . This leads to an approximation threshold for our problems.

**Theorem 2.** *For  $k \geq 2$ , there is a constant  $c > 0$  such that CR $k$  and CR $k$ ONE are inapproximable in polynomial time within a factor of  $c \ln n$  even on  $n$ -vertex bipartite graphs, unless P = NP.*

*Proof.* By Lemma 2 and Corollary 1, it follows that if there is an  $\alpha$ -approximation algorithm for CR2 then there is also an  $\alpha$ -approximation algorithm for SC. According to [17,1], under the assumption that P  $\neq$  NP, for a positive constant  $c$ , there is no  $c \ln |S|$ -approximation algorithm for a Set Cover instance (in which the family  $\mathcal{F}$  has size polynomial in  $|S|$ ). Thus, there is a positive constant  $c$  such that CR2 cannot be approximated within a factor of  $c \ln n$  on  $n$ -vertex bipartite graphs.  $\square$

## 4 Polynomial-Time Algorithms for Graphs with few $P_4$ 's

In this section we show how to obtain polynomial-time algorithms for CR2 and CR2ONE on  $(q, q-4)$ -graphs, for every fixed  $q$ . We recall that  $P_4$  denotes a path on 4 vertices, and that  $(q, q-4)$  is a graph in which every subset of at most  $q$  vertices induces at most  $q-4$  distinct  $P_4$ 's. Every  $n$ -vertex graph  $G$  is  $(q, q-4)$  for some  $q = q(G)$  (this can be determined in  $O(n^7)$  time [19]). For instance,  $q = 4$  for cographs and  $q = 5$  for  $P_4$ -sparse graphs.

These graphs have a nice recursive decomposition based on unions, joins, spiders and small separable p-components [2], all defined below.

Let  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$  be two vertex disjoint graphs. The *disjoint union* of  $G_1$  and  $G_2$  is the graph  $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$ . The *join* is the graph  $G_1 \vee G_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup \{uv : u \in V_1, v \in V_2\})$ .

A *spider* is a graph whose vertex set has a partition  $(R, K, S)$ , where  $K = \{k_1, \dots, k_p\}$  and  $S = \{s_1, \dots, s_p\}$ , for  $p \geq 2$ , are a clique and a stable set, respectively;  $s_i$  is adjacent to  $k_j$  if and only if  $i = j$  (a thin spider), or  $s_i$  is adjacent to  $k_j$  if and only if  $i \neq j$  (a thick spider); and every vertex of  $R$  is adjacent to each vertex of  $K$  and non-adjacent to each vertex of  $S$ .

We say that a graph is *p-connected* (path connected) if, for every bipartition of the vertex set, there is a crossing  $P_4$ , i.e. a  $P_4$  with vertices in both parts. A *separable p-component* is a maximal p-connected subgraph with a particular bipartition  $(H_1, H_2)$  such that every crossing  $P_4$   $wxyz$  satisfies  $x, y \in H_1$  and  $w, z \in H_2$ .

**Theorem 3.** [Primeval Decomposition [2]] *If  $G$  is a  $(q, q - 4)$ -graph, then one of the following holds:*

- (a)  *$G$  is the union or the join of two  $(q, q - 4)$ -graphs;*
- (b)  *$G$  is a spider  $(R, K, S)$  and  $G[R]$  is a  $(q, q - 4)$ -graph;*
- (c)  *$G$  contains a separable p-component  $H$ , with bipartition  $(H_1, H_2)$  and with  $|V(H)| \leq q$ , such that  $G - H$  is a  $(q, q - 4)$ -graph and every vertex of  $G - H$  is adjacent to every vertex of  $H_1$  and non-adjacent to every vertex of  $H_2$ ;*
- (d)  *$G$  has at most  $q$  vertices or  $V(G) = \emptyset$ .*

As a consequence, a  $(q, q - 4)$ -graph  $G$  can be decomposed by successively applying Theorem 3 as follows: If (a) holds, apply the theorem to each component of  $G$  or  $\overline{G}$  (operations disjoint union and join). If (b) holds, apply the theorem to  $G[R]$ . Finally, if (c) holds, then apply the theorem to  $G - H$ . This decomposition can be obtained in linear time [3,11]. Once we have it, problems CR2 and CR2ONE can be solved by composing the solution of the parts obtained in each case (a)-(d), as follows. From now on, let  $C$  be a 2-coloring with colors  $r$  and  $\bar{r}$ ,  $\{r, \bar{r}\} = \{1, 2\}$ ,  $G_{C,r}$  be the subgraph of  $G$  induced by  $r$  in  $C$ , and  $n_r(G, C) = |V(G_{C,r})|$ . Let  $\rho_2(G, C)$  and  $\rho_1(G, C, r)$  denote the optimal value of CR2 and CR2ONE, respectively.

**Lemma 3.** *Let  $G$  be the disjoint union of two disjoint graphs  $G_1$  and  $G_2$ . Then,  $\rho_1(G, C, r) = \min\{\rho_1(G_1, C, r) + n_r(G_2, C), \rho_1(G_2, C, r) + n_r(G_1, C)\}$ . If  $G_1$  or  $G_2$  is disconnected, then CR2 is infeasible; otherwise,  $\rho_2(G, C) = \min\{n_1(G_1, C) + n_2(G_2, C), n_2(G_1, C) + n_1(G_2, C)\}$ .*

*Proof.* Let  $(G, C, r)$  be an instance of CR2ONE. Clearly, in any solution, color  $r$  can appear only in  $G_1$  or  $G_2$ . If color  $r$  appears only in  $G_1$ , then we have to solve CR2ONE with instance  $(G_1, C, r)$  and recolor every vertex colored  $r$  in  $G_2$ . If color  $r$  appears only in  $G_2$ , then we have to solve CR2ONE with instance  $(G_2, C, r)$  and recolor every vertex colored  $r$  in  $G_1$ . This argument leads to  $\rho_1(G, C, r) = \min\{\rho_1(G_1, C, r) + n_r(G_2, C), \rho_1(G_2, C, r) + n_r(G_1, C)\}$ .

Now, let  $(G, C)$  be an instance of CR2. If  $G_1$  or  $G_2$  is disconnected, then it is easy to see that there is no possible convex recoloring with two colors. If  $G_1$  and  $G_2$  are both connected, then there is only two possible convex recolorings: (a) color 1 appears only in  $G_1$  and color 2 appears only in  $G_2$ , or (b) color 1 appears only in  $G_2$  and color 2 appears only in  $G_1$ . This argument leads to  $\rho_2(G, C) = \min\{n_1(G_1, C) + n_2(G_2, C), n_2(G_1, C) + n_1(G_2, C)\}$ .

**Lemma 4.** *Let  $G$  be the join of  $G_1$  and  $G_2$ . If  $G_{C,r}$  is disconnected, then  $\rho_1(G, C, r) = 1$ ; otherwise,  $\rho_1(G, C, r) = 0$ . If  $V(G_1) = \{v\}$ , then  $\rho_2(G, C) = \min\{\rho_1(G_2, C, \overline{C(v)}), \rho_1(G_2, C, C(v)) + 1\}$ . If  $G_1$  and  $G_2$  have at least 2 vertices*

each, then  $\rho_2(G, C) = d(G, C) := \rho_1(G, C, 1) + \rho_1(G, C, 2)$ , if  $d(G, C) \neq 2$  or  $d(G - \{v\}, C) \neq 1$  for all  $v \in V(G)$ ; and  $\rho_2(G, C) = 1$ , otherwise.

*Proof.* Let  $(G, C, r)$  be an instance of CR2ONE. If  $G_{C,r}$  is disconnected,  $G_1$  or  $G_2$  does not contain a vertex colored  $r$ . Assume it is the case of  $G_1$ . Recoloring one vertex of  $G_1$  (to color  $r$ ) will connect the subgraph induced by color  $r$ .

Let  $(G, C)$  be an instance of CR2. Suppose that both  $G_1$  and  $G_2$  have more than one vertex. The case  $d(G, C) \leq 1$  is similar to CR2ONE, so that  $\rho_2(G, C) = d(G, C)$ . Now suppose that  $d(G, C) = 2$ . Without loss of generality, we can assume that color  $i$  only appears in  $G_i$ , for  $i = 1, 2$ . If  $d(G - \{v\}, C) = 1$  for some  $v \in V(G)$ , we just need to switch the color of  $v$  to get a solution. Otherwise, one vertex from  $G_1$  and one vertex from  $G_2$  must be recolored. Finally, suppose that  $V(G_1) = \{v\}$ . Then, a convex recoloring of  $G$  that assigns color  $r$  to  $v$  is a recoloring of  $G_2$  where color  $\bar{r}$  induces a connected subgraph. This leads to  $\rho_2(G, C) = \min\{\rho_1(G_2, C, \overline{C(v)}), \rho_1(G_2, C, C(v)) + 1\}$ .

For a thick spider with  $|K| = 2$ , which is equivalent to a thin spider, the analysis is very particular and will be omitted. We focus on  $|K| \geq 3$ . In what follows, for simplicity, we may say that a *color  $c$  is connected*, meaning that the subgraph induced by the vertices with color  $c$  is connected.

**Lemma 5.** *Let  $G$  be a thick spider with  $|K| \geq 3$ . If  $S$  is colored  $r$  and  $K$  is colored  $\bar{r}$ ,  $\rho_1(G, C, r) = 2$ . Otherwise,  $\rho_1(G, C, r)$  is 0 or 1, depending on whether  $G_{C,r}$  is connected or not. Moreover,  $\rho_2(G, C) = \max\{\rho_1(G, C, 1), \rho_1(G, C, 2)\}$ .*

*Proof.* First, note that a color  $c$  is connected if  $n_c(K, C) \geq 2$ . Then, since  $|K| \geq 3$  and we have only two possible colors, there is at least one connected color. Assume that only one color is disconnected, let us say  $r$ ; otherwise, both colors are already connected. Then,  $\rho_2(G, C) \geq 1$  and  $\rho_1(G, C, r) \geq 1$ . Moreover,  $n_r(K, C) \leq 1$ . If  $n_r(K, C) = 1$ , then  $C(k_i) = C(s_i) = r$ , for some  $i$  (as  $r$  is disconnected). Recoloring  $s_i$  is enough to get a convex coloring. If  $n_r(K, C) = 0$ , we consider two subcases. If there is a vertex  $s_i$  colored  $\bar{r}$ , we can recolor  $k_i$  to get a convex recoloring. Otherwise, every vertex of  $S$  is colored  $r$  and every vertex of  $K$  is colored  $\bar{r}$ . Then, we must recolor at least two vertices to connect the vertices of color  $r$ . Actually, we could recolor  $s_i$  and  $k_i$  to connect both colors. These cases show the desired results.

**Lemma 6.** *Let  $G$  be a thin spider with  $|K| \geq 3$ , and  $S_r = \{s_i \in S : C(s_i) = r, C(k_i) = \bar{r}\}$ . Then,  $\rho_1(G, C, r) = |S_r|$  and  $\rho_2(G, C) = |S_1| + |S_2|$  except in two cases: (i)  $S \cup K$  is colored  $\bar{r}$ :  $\rho_1(G, C, r) = 1$  if  $G_{C,r}$  is disconnected, and  $\rho_1(G, C, r) = 0$ , otherwise;  $\rho_2(G, C) = \min\{2, \rho_1(G[R], C, r)\}$ ; (ii)  $K \cup R$  is colored  $\bar{r}$ :  $\rho_1(G, C, r) = \rho_2(G, C) = \max\{0, |S_r| - 1\}$ .*

*Proof.* First, suppose that both  $r$  and  $\bar{r}$  appear in  $K$ . If  $S_r = \emptyset$ ,  $G_{C,r}$  is connected. Otherwise, it suffices to recolor the vertices in  $S_r$  to connect color  $r$ . However, recoloring less than  $|S_r|$  vertices will keep the original color of at least one pair  $s_i, k_i$ , with  $s_i \in S_r$ , so that color  $r$  is still disconnected. Extending this argumentation to  $\bar{r}$ , we can show that all the vertices in  $S_r$  and  $S_{\bar{r}}$  must be recolored to connect both colors. Therefore,  $\rho_1(G, C, r) = |S_r|$  and  $\rho_2(G, C) = |S_1| + |S_2|$ .

Now, suppose that  $K$  is monochromatic. If it is colored  $r$ , then  $\rho_1(G, C, r) = 0 = |S_r|$ . If it is colored  $\bar{r}$ , we consider three subcases:

1.  $S$  is colored  $\bar{r}$ : If disconnected, the subgraph of  $G$  induced by color  $r$  can be made connected by switching the color of a vertex  $k_i \in K$ . Thus, we get the desired value for  $\rho_1(G, C, r)$ .
2. Color  $r$  appears in  $S$  and  $R$  is colored  $\bar{r}$ : If  $|S_r| \leq 1$ , then  $G_{C,r}$  is connected. Otherwise, switching the color of all but one vertex of  $S_r$  connects color  $r$ . On the other hand, recoloring less than  $|S_r| - 1$  vertices will keep the original color of at least two pairs  $s_i, k_i$ , with  $s_i \in S_r$ , so that color  $r$  is still disconnected. Hence,  $\rho_1(G, C, r) = \max\{0, |S_r| - 1\}$ .
3. Color  $r$  appears in  $S$  and  $R$ : Besides the vertices recolored in subcase (2), we need to recolor the vertex in  $K$  whose neighbor in  $S_r$  has kept its color. The argumentation is similar, leading to  $\rho_1(G, C, r) = |S_r| = |S_r| + |S_{\bar{r}}|$ .

To determine  $\rho_2(G, C)$ , we can assume that  $K$  is colored  $\bar{r}$  so as to consider subcases (1)-(3). In subcases (2) and (3), note that color  $\bar{r}$  is kept connected, so that  $\rho_2(G, C) = \rho_1(G, C, r)$ . In subcase (2), we can get the two colors connected by either recoloring  $k_i$  and  $s_i$  or recoloring  $\rho_1(G[R], C, r)$  vertices in  $G[R]$ . Again, we obtain the desired expression.

Finally, we deal with case (c) of Theorem 3.

**Lemma 7.** *Let  $G$  contain a separable  $p$ -component  $H = H_1 \cup H_2$  with less than  $q$  vertices such that  $G \setminus H$  is complete to  $H_1$  and anti-complete to  $H_2$ . Let  $\mathcal{C}_r(H)$  be the set of recolorings  $\Gamma$  of  $H$  where  $H_{\Gamma,r}$  is connected or empty, or each connected component contains a vertex in  $H_1$ . Let  $\alpha(\Gamma)$  be the number of recolored vertices of  $H$  in  $\Gamma$ , and  $\beta_r(\Gamma)$  be equal to  $\rho_1(G - H, C, r)$ , if  $V(H_{\Gamma,r}) = \emptyset$ ;  $n_r(G - H, C)$ , if  $\emptyset \neq V(H_{\Gamma,r}) \subseteq H_2$ ; 1, if  $H_{\Gamma,r}$  is disconnected and  $n_r(G \setminus H, C) = 0$ ; 0, otherwise. Then,  $\rho_1(G, C, r) = \min\{\alpha(\Gamma) + \beta_r(\Gamma) : \Gamma \in \mathcal{C}_r(H)\}$  and  $\rho_2(G, C) = \min\{\alpha(\Gamma) + \max\{\beta_1(\Gamma), \beta_2(\Gamma)\} : \Gamma \in \mathcal{C}_1(H) \cap \mathcal{C}_2(H)\}$ .*

*Proof.* For CR2ONE, we have to show that  $\mathcal{C}_r(H)$  comprises exactly the recolorings of  $H$  that can be extended to recoloring of  $G$  connecting  $r$ , and that  $\beta_r(H)$  is the minimum number of recolorings in  $G - H$  to get this extension. First, let  $\Gamma$  be a recoloring of  $H$  not belonging to  $\mathcal{C}_r(H)$ . Then,  $H_{\Gamma,r}$  is disconnected, being one of its connected components included in  $H_2$ . Since  $H_2$  is anti-complete to  $G - H$ , we cannot connect color  $r$  with any recoloring of  $G - H$ .

Now, let  $\Gamma \in \mathcal{C}_r(H)$ . We consider four cases. If  $H_{\Gamma,r}$  is empty, we trivially have that  $\rho_1(G - H, C, r)$  is the minimum number of recolorings in  $G - H$  to extend  $\Gamma$  to a recoloring of  $G$  connecting  $r$ . If  $H_{\Gamma,r}$  is nonempty, connected and included in  $H_2$ , which is anti-complete to  $G - H$ , the only way to connect  $r$  in  $G$  is by recoloring every vertex in  $G - H$  colored  $r$  in  $C$ . If  $H_{\Gamma,r}$  is nonempty, connected and contains a vertex in  $H_1$ , color  $r$  is already connected in  $G$ , provided that  $H_1$  is complete to  $G - H$ . In the remaining case, each of the (at least two) connected components of  $H_{\Gamma,r}$  has a vertex in  $H_1$ . We just need a vertex colored  $r$  in  $G - H$  to get color  $r$  connected. If there is no such a vertex colored  $r$  by  $C$ , an additional recoloring is necessary.

For CR2, we can apply the above argument to both  $r$  and  $\bar{r}$ . The first part of the proof shows that  $\mathcal{C}(H) = \mathcal{C}_r(C) \cap \mathcal{C}_{\bar{r}}(C)$  comprises exactly the recolorings of  $H$  that can be extended to a convex recoloring of  $G$ . Then, for each  $\Gamma \in \mathcal{C}(H)$ , we can use the second part of the proof to determine  $\beta(\Gamma) = \max\{\beta_r(\Gamma), \beta_{\bar{r}}(\Gamma)\}$  and see that  $\beta(\Gamma)$  is the minimum number of recolorings in  $G - H$  to extend  $\Gamma$  to a convex recoloring of  $G$ .

## 5 Concluding Remarks

The convex recoloring problem, in its simpler form (the unweighted case), as we have stated here, is an easy to state combinatorial optimization problem, which seems to be tractable for some simple classes of graphs. It is, thus, rather unexpected that for grid graphs and only two colors the corresponding problem is already NP-hard. It would be interesting to design an approximation algorithm for CR2 on this class of graphs. Another related question is whether CR2 (or CR $k$ ) on graphs with maximum degree 3 can be easily solved or not.

As we showed in this paper, CR $k$  and CR $k$ ONE (and CR) cannot be approximated within a logarithmic factor, unless P=NP. This is a strong inapproximability result for arbitrary graphs (even with only 2 colors). It is a challenging problem to show other inapproximability results for more special classes of graphs or find larger classes of graphs for which a polynomial or a constant approximation algorithm for these problems can be designed. To our knowledge, the largest class for which a constant approximation algorithm (for CR) has been designed is the class of graphs with bounded treewidth [9].

## References

1. Alon, N., Moshkovitz, D., Safra, S.: Algorithmic construction of sets for  $k$ -restrictions. *ACM Transactions on Algorithms* 2, 153–177 (2006)
2. Babel, L., Olariu, S.: On the structure of graphs with few  $P_4$ 's. *Discrete Appl. Math.* 84, 1–13 (1998)
3. Baumann, S.: A linear algorithm for the homogeneous decomposition of graphs, Report No. M-9615, Zentrum für Mathematik, Technische Universität München (1996)
4. Bar-Yehuda, R., Feldman, I., Rawitz, D.: Improved approximation algorithm for convex recoloring of trees. *Theor. Comp. Sys.* 43, 3–18 (2008)
5. Bodlaender, H.L., Fellows, M.R., Langston, M.A., Ragan, M.A., Rosamond, F.A., Weyer, M.: Quadratic kernelization for convex recoloring of trees. *Algorithmica* 61(2), 362–388 (2011)
6. Campêlo, M.B., Lima, K.R., Moura, P.F.S., Wakabayashi, Y.: Polyhedral studies on the convex recoloring problem (2012), accepted to VII Latin-American Algorithms, Graphs and Optimization Symposium (2013)
7. Chor, B., Fellows, M., Ragan, M.A., Razgon, I., Rosamond, F., Snir, S.: Connected coloring completion for general graphs: Algorithms and complexity. In: Lin, G. (ed.) COCOON 2007. LNCS, vol. 4598, pp. 75–85. Springer, Heidelberg (2007)
8. Garey, M.R., Johnson, D.S.: The rectilinear steiner tree problem in NP-complete. *SIAM Journal of Applied Mathematics* 32, 826–834 (1977)

9. Kammer, F., Tholey, T.: The complexity of minimum convex coloring. *Discrete Appl. Math.* 160, 810–833 (2012)
10. Kanj, I.A., Kratsch, D.: Convex recoloring revisited: Complexity and exact algorithms. In: Ngo, H.Q. (ed.) COCOON 2009. LNCS, vol. 5609, pp. 388–397. Springer, Heidelberg (2009)
11. Jamison, B., Olariu, S.: A tree representation for  $P_4$ -sparse graphs. *Discrete Appl. Math.* 35, 115–129 (1992)
12. Lima, K.R., Wakabayashi, Y.: Convex recoloring of paths. *Electronic Notes in Discrete Mathematics* 37, 165–170 (2011)
13. Moran, S., Snir, S.: Efficient approximation of convex recolorings. *J. Comput. Syst. Sci.* 73, 1078–1089 (2007)
14. Moran, S., Snir, S.: Convex recolorings of strings and trees: Definitions, hardness results and algorithms. *J. Comput. Syst. Sci.* 74, 850–869 (2008)
15. Moran, S., Snir, S., Sung, W.-K.: Partial convex recolorings of trees and galled networks: tight upper and lower bounds. *ACM Trans. Algorithms* 7 (2011)
16. Ponta, O., Hüffner, F., Niedermeier, R.: Speeding up dynamic programming for some NP-hard graph recoloring problems. In: Agrawal, M., Du, D.-Z., Duan, Z., Li, A. (eds.) TAMC 2008. LNCS, vol. 4978, pp. 490–501. Springer, Heidelberg (2008)
17. Raz, R., Safra, S.: A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In: Proc. of the 29th Annual ACM Symposium on Theory of Computing, pp. 475–484 (1997)
18. Razgon, I.: A  $2^{O(k)}\text{poly}(n)$  algorithm for the parameterized convex recoloring problem. *Inform. Process. Lett.* 104(2), 53–58 (2007)
19. Sales, C.L., Maia, A.K., Martins, N., Sampaio, R.M.: Restricted Coloring Problems on graphs with few  $P_4$ 's. *Annals of Operations Research* (to appear)

# 2-connecting Outerplanar Graphs without Blowing Up the Pathwidth

Jasine Babu<sup>1</sup>, Manu Basavaraju<sup>2</sup>, Sunil Chandran Leela<sup>1</sup>,  
and Deepak Rajendraprasad<sup>1</sup>

<sup>1</sup> Indian Institute of Science, Bangalore 560012, India

<sup>2</sup> The Institute of Mathematical Sciences, Chennai 600113, India  
`{jasine,sunil,deepakr}@csa.iisc.ernet.in, manub@imsc.res.in`

**Abstract.** Given a connected outerplanar graph  $G$  with pathwidth  $p$ , we give an algorithm to add edges to  $G$  to get a supergraph of  $G$ , which is 2-vertex-connected, outerplanar and of pathwidth  $O(p)$ . As a consequence, we get a constant factor approximation algorithm to compute a straight line planar drawing of any outerplanar graph, with its vertices placed on a two dimensional grid of minimum height. This settles an open problem raised by Biedl [3].

**Keywords:** Pathwidth, Outerplanar Graph, Bi-connected.

## 1 Introduction

A graph  $G(V, E)$  is outerplanar, if it has a planar embedding with all its vertices lying on the outer face. Computing planar straight line drawings of planar graphs with vertices placed on a two dimensional grid, is a well known problem in graph drawing. The height of a grid is defined as the smaller of the two dimensions of the grid. If  $G$  has a planar straight line drawing, with its vertices placed on a two dimensional grid of height  $h$ , then we call it a planar drawing of  $G$  of height  $h$ . It is known that any planar graph on  $n$  vertices can be drawn on an  $(n - 1) \times (n - 1)$  sized grid [11].

We use  $\text{pw}(G)$  to denote the pathwidth of a graph  $G$ . Pathwidth is a structural parameter of graphs, which is widely used in graph drawing and layout problems [3,5,13]. The study of pathwidth, in the context of graph drawings, was initiated by Dujmovic et al. [5]. It is known that any planar graph that has a planar drawing of height  $h$  has pathwidth at most  $h$  [13]. However, there exist planar graphs of constant pathwidth but requiring  $\Omega(n)$  height in any planar drawing [2]. In the special case of trees, Suderman [13] showed that any tree  $T$  has a planar drawing of height at most  $3\text{ pw}(T) - 1$ . Biedl [3] considered the same problem for the bigger class of outerplanar graphs. For any bi-connected outerplanar graph  $G$ , Biedl [3] obtained an algorithm to compute a planar drawing of  $G$  of height at most  $4\text{ pw}(G) - 3$ . Since it is known that pathwidth is a lower bound for the height of the drawing [13], the algorithm given by Biedl [3] is a 4-factor approximation algorithm for the problem, for any bi-connected outerplanar graph. The method

in Biedl [3] is to add edges to the bi-connected outerplanar graph  $G$  to make it a maximal outerplanar graph  $H$  and then draw  $H$  on a grid of height  $4\text{pw}(G) - 3$ . The same method would give a constant factor approximation algorithm for arbitrary outerplanar graphs, if it is possible to add edges to any arbitrary outerplanar graph  $G$  to obtain a bi-connected outerplanar graph  $G'$  such that  $\text{pw}(G') = O(\text{pw}(G))$ . This was an open problem in Biedl [3].

In this paper, we give an algorithm to augment a connected outerplanar graph  $G$  of pathwidth  $p$  by adding edges so that the resultant graph is a bi-connected outerplanar graph of pathwidth  $O(p)$ . Notice that, the non-triviality lies in the fact that  $G'$  has to be maintained outerplanar. If we relax this condition, the problem becomes very easy. It is easy to verify that the supergraph  $G'$  of  $G$ , obtained by making any two vertices of  $G$  adjacent to each other and to every other vertex in the graph, is bi-connected and has pathwidth at most  $\text{pw}(G) + 2$ . The problem of augmenting outerplanar graphs to make them bi-connected, while maintaining the outerplanarity and optimizing some other properties, like number of edges added [6,9], have been investigated previously.

## 2 Background

A *tree decomposition* of a graph  $G(V, E)$  [10] is a pair  $(T, \mathcal{X})$ , where  $T$  is a tree and  $\mathcal{X} = (X_t : t \in V(T))$  is a family of subsets of  $V(G)$ , such that:

1.  $\bigcup(X_t : t \in V(T)) = V(G)$ .
2. For every edge  $e$  of  $G$  there exists  $t \in V(T)$  such that  $e$  has both its end points in  $X_t$ .
3. For  $t, t', t'' \in V(T)$ , if  $t'$  is on the path of  $T$  between  $t$  and  $t''$  then,  $X_t \cap X_{t''} \subseteq X_{t'}$ .

The width of the tree decomposition is  $\max_{t \in V(T)}(|X_t| - 1)$ . Each  $X_t \in \mathcal{X}$  is referred to as a bag in the tree decomposition. The graph  $G$  has *treewidth*  $w$  if  $w$  is the minimum such that  $G$  has a tree decomposition of width  $w$ . A *path decomposition*  $(P, \mathcal{X})$  of a graph  $G$  is a tree decomposition of  $G$  with the additional property that the tree  $P$  is a path. The width of the path decomposition is  $\max_{t \in V(P)}(|X_t| - 1)$ . The graph  $G$  has *pathwidth*  $w$  if  $w$  is the minimum such that  $G$  has a path decomposition of width  $w$ .

Without loss of generality we can assume that, in any path decomposition  $(P, \mathcal{X})$  of  $G$ , the vertices of the path  $P$  are labeled as  $1, 2, \dots$ , in the order in which they appear in  $P$ . Accordingly, the bags in  $\mathcal{X}$  also get indexed as  $1, 2, \dots$ . For each vertex  $v \in V(G)$ , define  $\text{FirstIndex}_{\mathcal{X}}(v) = \min\{i \mid X_i \in \mathcal{X} \text{ contains } v\}$ ,  $\text{LastIndex}_{\mathcal{X}}(v) = \max\{i \mid X_i \in \mathcal{X} \text{ contains } v\}$  and  $\text{Range}_{\mathcal{X}}(v) = [\text{FirstIndex}_{\mathcal{X}}(v), \text{LastIndex}_{\mathcal{X}}(v)]$ . By the definition of a path decomposition, if  $t \in \text{Range}_{\mathcal{X}}(v)$ , then  $v \in X_t$ . If  $v_1$  and  $v_2$  are two distinct vertices, define  $\text{Gap}_{\mathcal{X}}(v_1, v_2)$  as follows:

- If  $\text{Range}_{\mathcal{X}}(v_1) \cap \text{Range}_{\mathcal{X}}(v_2) \neq \emptyset$ , then  $\text{Gap}_{\mathcal{X}}(v_1, v_2) = \emptyset$ .
- If  $\text{LastIndex}_{\mathcal{X}}(v_1) < \text{FirstIndex}_{\mathcal{X}}(v_2)$ , then  

$$\text{Gap}_{\mathcal{X}}(v_1, v_2) = [\text{LastIndex}_{\mathcal{X}}(v_1) + 1, \text{FirstIndex}_{\mathcal{X}}(v_2)].$$
- If  $\text{LastIndex}_{\mathcal{X}}(v_2) < \text{FirstIndex}_{\mathcal{X}}(v_1)$ , then  

$$\text{Gap}_{\mathcal{X}}(v_1, v_2) = [\text{LastIndex}_{\mathcal{X}}(v_2) + 1, \text{FirstIndex}_{\mathcal{X}}(v_1)].$$

The motivation for this definition is the following. Suppose  $(P, \mathcal{X})$  is a path decomposition of a graph  $G$  and  $v_1$  and  $v_2$  are two non-adjacent vertices of  $G$ . If we add a new edge between  $v_1$  and  $v_2$ , a natural way to modify the path decomposition to reflect this edge addition is the following. If  $\text{Gap}_{\mathcal{X}}(v_1, v_2) = \emptyset$ , there is an  $X_t \in \mathcal{X}$ , which contains  $v_1$  and  $v_2$  together and hence, we need not modify the path decomposition. If  $\text{LastIndex}_{\mathcal{X}}(v_1) < \text{FirstIndex}_{\mathcal{X}}(v_2)$ , we insert  $v_1$  into all  $X_t \in \mathcal{X}$ , such that  $t \in \text{Gap}_{\mathcal{X}}(v_1, v_2)$ . On the other hand, if  $\text{LastIndex}_{\mathcal{X}}(v_2) < \text{FirstIndex}_{\mathcal{X}}(v_1)$ , we insert  $v_2$  to all  $X_t \in \mathcal{X}$ , such that  $t \in \text{Gap}_{\mathcal{X}}(v_1, v_2)$ . It is clear from the definition of  $\text{Gap}_{\mathcal{X}}(v_1, v_2)$ , that this procedure gives a path decomposition of the new graph. Whenever we add an edge  $(v_1, v_2)$ , we stick to this procedure to update the path decomposition.

A *block* of a graph  $G$  is a maximal connected subgraph of  $G$  without a cut vertex. Every block of a connected graph  $G$  is thus either a single edge which is a bridge in  $G$ , or a maximal bi-connected subgraph of  $G$ . If a block of  $G$  is not a single edge, we call it as a non-trivial block of  $G$ . Given a connected outerplanar graph  $G$ , we define a rooted tree  $T$  (hereafter referred to as the *rooted block tree* of  $G$ ) as follows. The vertices of  $T$  are the blocks of  $G$  and the root of  $T$  is an arbitrary block of  $G$  which contains at least one non-cut vertex (it is easy to see that such a block always exists). Two vertices  $B_i$  and  $B_j$  of  $T$  are adjacent if the blocks  $B_i$  and  $B_j$  share a cut vertex in  $G$ . It is easy to see that  $T$ , as defined above, is a tree. In our discussions, we restrict ourselves to a fixed rooted block tree of  $G$ . If block  $B_i$  is a child block of block  $B_j$  in the rooted block tree of  $G$ , and they share a cut vertex  $x$ , we say that  $B_i$  is a child block of  $B_j$  at  $x$ .

It is known that every bi-connected outerplanar graph has a unique Hamiltonian cycle [14]. Though the Hamiltonian cycle of a bi-connected block of  $G$  can be traversed either clockwise or anticlockwise, let us fix one of these orderings, so that the successor and predecessor of each vertex in the Hamiltonian cycle of the block is fixed. We call this order as the clockwise order. Consider a non-root block  $B_i$  of  $G$  such that  $B_i$  is a child block of its parent, at the cut vertex  $x$ . If  $B_i$  is a non-trivial block and  $y_i$  and  $y'_i$  respectively be the predecessor and successor of  $x$  in the Hamiltonian cycle of  $B_i$ , we call  $y_i$  as the last vertex of  $B_i$  and  $y'_i$  as the first vertex of  $B_i$ . If  $B_i$  is a trivial block, the neighbor of  $x$  in  $B_i$  is regarded as both the first vertex and the last vertex of  $B_i$ . By the term path we always mean a simple path, i.e., a path in which no vertex repeats.

### 3 An Overview of Our Method

Given a connected outerplanar graph  $G(V, E)$  of pathwidth  $p$ , our algorithm will produce a bi-connected outerplanar graph  $G''(V, E'')$  of pathwidth  $O(p)$ , where  $E \subseteq E''$ . Our algorithm proceeds in three stages.

(1) We use a modified version of the algorithm proposed by Govindan et al. [7] to obtain a *nice tree decomposition* (defined in Section 4) of  $G$ . Using this nice tree decomposition of  $G$ , we construct a special path decomposition of  $G$  of width at most  $4p + 3$ .

(2) For each cut vertex  $x$  of  $G$ , we define an ordering among the child blocks attached through  $x$  to their parent block. To define this ordering, we use the

special path decomposition constructed in the first stage. This ordering helps us in constructing an outerplanar supergraph  $G'(V, E')$  whose pathwidth is at most  $8p + 7$ , and for every cut vertex  $x$  in  $G'$ ,  $G' \setminus x$  has exactly two components. The properties of the special path decomposition of  $G$  obtained in the first stage is crucially used in our argument to bound the width of the path decomposition of  $G'$ , produced in the second stage.

(3) We bi-connect  $G'$  to construct  $G''(V, E'')$ , using a straightforward algorithm. As a by-product, this algorithm also gives us a surjective mapping from the cut vertices of  $G'$  to the edges in  $E'' \setminus E'$ . We give a counting argument based on this mapping and some basic properties of path decompositions to show that the width of the path decomposition of  $G''$  produced in the third stage is at most  $16p + 15$ .

## 4 Stage 1: Construct a Nice Path Decomposition of $G$

In this section, we construct a *nice tree decomposition* of the outerplanar graph  $G$  and then use it to construct a *nice path decomposition* of  $G$ . We begin by giving the definition of a nice tree decomposition.

Given an outerplanar graph  $G$ , Govindan et al. [7, Section 2] gave a linear time algorithm to construct a width 2 tree decomposition  $(T, \mathcal{Y})$  of  $G$  where  $\mathcal{Y} = (Y_t : t \in V(T))$ , with the following special properties:

1. There is a bijective mapping  $b$  from  $V(G)$  to  $V(T)$  such that  $v \in Y_{b(v)}$ . (Hereafter, for any  $v \in V(G)$ , while referring to the vertex  $b(v)$  of  $T$ , we just call it as vertex  $v$  of  $T$ .)
2. If  $B_i$  is a child block of  $B_j$  at a cut vertex  $x$ , the induced subgraph  $T'$  of  $T$  on the vertex set  $V(B_i \setminus x)$  is a spanning tree of  $B_i \setminus x$  and  $(T', \mathcal{Y}')$  where  $\mathcal{Y}' = (Y_t : t \in V(T'))$  gives a tree decomposition of  $B_i$ .

**Definition 1 (Nice tree decomposition of an outerplanar graph  $G$ ).** A tree decomposition  $(T, \mathcal{Y})$  of  $G$ , where  $\mathcal{Y} = (Y_t : t \in V(T))$  having properties 1 and 2 above, together with the following additional property, is called a nice tree decomposition of  $G$ .

3. If  $y_i$  and  $y'_i$  are respectively the last and first vertices of a non-root, non-trivial block  $B_i$ , then the bag  $Y_{y_i} \in \mathcal{Y}$  contains both  $y_i$  and  $y'_i$ .

In the discussion that follows, we will show that any outerplanar graph  $G$  has a nice tree decomposition  $(T, \mathcal{Y})$  of width at most 3. Initialize  $(T, \mathcal{Y})$  to be the tree decomposition of  $G$ , constructed using the method proposed by Govindan et al. [7], satisfying properties 1 and 2 of nice tree decompositions. We need to modify  $(T, \mathcal{Y})$  in such a way that, it satisfies property 3 as well.

For every non-root, non-trivial block  $B_i$  of  $G$ , do the following. If  $y_i$  and  $y'_i$  are respectively the last and first vertices of  $B_i$ , then, for each  $t \in V(B_i \setminus x)$ , we insert  $y'_i$  to  $Y_t$ , if it is not already present in  $Y_t$  and we call  $y'_i$  as a *propagated* vertex.

*Claim 1.* After the modification,  $(T, \mathcal{Y})$  remains as a tree decomposition of  $G$ .

*Claim 2.* After the modification,  $(T, \mathcal{Y})$  becomes a nice tree decomposition of  $G$  of width at most 3.

The proofs of the claims above are included in the full version [1]. From these claims, we can conclude the following.

**Lemma 1.** *Every outerplanar graph  $G$  has a nice tree decomposition  $(T, \mathcal{Y})$  of width 3, constructable in polynomial time.*

**Definition 2 (Nice path decomposition of an outerplanar graph).** *Let  $(\mathcal{P}, \mathcal{X})$  be a path decomposition of an outerplanar graph  $G$ . If, for every non-root non-trivial block  $B_i$ , there is a bag  $X_t \in \mathcal{X}$  containing both the first and last vertices of  $B_i$  together, then  $(\mathcal{P}, \mathcal{X})$  is called a nice path decomposition of  $G$ .*

**Lemma 2.** *Let  $G$  be outerplanar with  $\text{pw}(G) = p$ . A nice path decomposition  $(\mathcal{P}, \mathcal{X})$  of  $G$ , of width at most  $4p + 3$ , is constructable in polynomial time.*

*Proof.* Let  $(\mathcal{T}, \mathcal{Y})$  be a nice tree decomposition of  $G$  of width 3, obtained using Lemma 1. Obtain an optimal path decomposition  $(\mathcal{P}_T, \mathcal{X}_T)$  of the tree  $\mathcal{T}$  in polynomial time, using a standard algorithm (See [12]). Since  $\mathcal{T}$  is a spanning tree of  $G$ , the pathwidth of  $\mathcal{T}$  is at most that of  $G$ . Therefore, the width of the path decomposition  $(\mathcal{P}_T, \mathcal{X}_T)$  is at most  $p$ ; i.e. there are at most  $p + 1$  vertices of  $\mathcal{T}$  in each bag  $X_{T_i} \in \mathcal{X}_T$ .

Let  $\mathcal{P} = \mathcal{P}_T$  and for each  $X_{T_i} \in \mathcal{X}_T$ , we define  $X_i = \bigcup_{v_T \in X_{T_i}} Y_{v_T}$ . Clearly,  $(\mathcal{P}, \mathcal{X})$ , with  $\mathcal{X} = (X_1, \dots, X_{|V(\mathcal{P}_T)|})$ , is a path decomposition of  $G$  (See [7]). The width of this path decomposition is at most  $4(p + 1) - 1 = 4p + 3$ , since  $|Y_{v_T}| \leq 4$ , for each bag  $Y_{v_T} \in \mathcal{Y}$  and  $|X_{T_i}| \leq p + 1$ , for each bag  $X_{T_i} \in \mathcal{X}_T$ .

Let  $B_i$  be a non-root, non-trivial block in  $G$  and  $y_i$  and  $y'_i$  respectively be the first and last vertices of  $B_i$ . Since  $y_i$  is a vertex of the tree  $\mathcal{T}$ , there is some bag  $X_{T_j} \in \mathcal{X}_T$ , containing  $y_i$ . The bag  $Y_{y_i} \in \mathcal{Y}$  contains both  $y_i$  and  $y'_i$ , since  $(\mathcal{T}, \mathcal{Y})$  is a nice tree decomposition of  $G$ . It follows from the definition of  $X_j$ , that  $X_j \in \mathcal{X}$  contains both  $y_i$  and  $y'_i$ . Therefore,  $(\mathcal{P}, \mathcal{X})$  is a nice path decomposition of  $G$ .  $\square$

## 5 Edge Addition without Spoiling the Outerplanarity

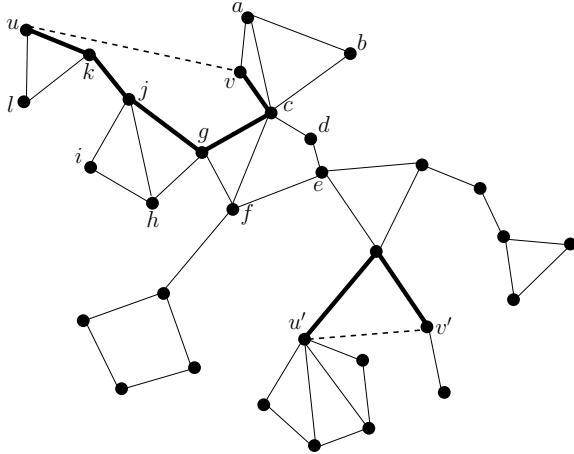
In this section, we give two technical lemmas, which will be later used to prove that the intermediate graph  $G'$  obtained in Stage 2 and the bi-connected graph  $G''$  obtained in Stage 3 are outerplanar. These lemmas are easy to visualize (See Fig 1). The proofs are included in the full version [1].

**Lemma 3.** *Let  $G(V, E)$  be a connected outerplanar graph. Let  $u$  and  $v$  be two distinct non-adjacent vertices in  $G$  and let  $P = (u = x_0, x_1, x_2, \dots, x_k, x_{k+1} = v)$  where  $k \geq 1$  be a path in  $G$  such that:*

*$P$  shares at most one edge with any block of  $G$ .*

*For  $0 \leq i \leq k$ , if the block containing the edge  $(x_i, x_{i+1})$  is non-trivial, then  $x_{i+1}$  is the successor of  $x_i$  in the Hamiltonian cycle of that block.*

*Then the graph  $G'(V, E')$ , where  $E' = E \cup \{(u, v)\}$ , is outerplanar.*



**Fig. 1.** The path between  $u$  and  $v$  and the path between  $u'$  and  $v'$  (shown in thick edges) satisfy the conditions stated in Lemma 3. By adding any one of the dotted edges  $(u, v)$  or  $(u', v')$ , the graph remains outerplanar. When the edge  $(u, v)$  is added,  $u, v, a, b, c, d, e, f, g, h, i, j, k, l, u$  is the Hamiltonian cycle of the new block formed.

The following lemma explains the effect of the addition of an edge  $(u, v)$  as mentioned in Lemma 3, to the block structure and the Hamiltonian cycle of each block. Assume that for  $0 \leq i \leq k$ , the edge  $(x_i, x_{i+1})$  belongs to the block  $B_i$ .

- Lemma 4.**
1. Other than the blocks  $B_0$  to  $B_k$  of  $G$  merging together to form a new block  $B'$  of  $G'$ , blocks in  $G$  and  $G'$  are the same.
  2. Vertices in blocks  $B_0$  to  $B_k$ , except  $x_i$ ,  $0 \leq i \leq k+1$ , retains their successor and predecessor in the Hamiltonian cycle of  $B'$  same as it was in its respective block's Hamiltonian cycle in  $G$ .
  3. Each  $x_i$ ,  $0 \leq i \leq k$ , retains its Hamiltonian cycle predecessor in  $B'$  same as it was in the block  $B_i$  of  $G$  and each  $x_i$ ,  $1 \leq i \leq k+1$ , retains its Hamiltonian cycle successor in  $B'$  same as in the block  $B_{i-1}$  of  $G$ .

## 6 Stage 2: Construction of $G'$ and Its Path Decomposition

For each cut vertex  $x$  of  $G$ , we define an ordering among the child blocks attached through  $x$  to their parent block, using the nice path decomposition  $(\mathcal{P}, \mathcal{X})$  of  $G$  obtained using Lemma 2. This ordering is then used in defining a supergraph  $G'(V, E')$  of  $G$  such that for every cut vertex  $x$  in  $G'$ ,  $G' \setminus x$  has exactly two components. Using repeated applications of Lemma 3, we then show that  $G'$  is outerplanar. We extend the path decomposition  $(\mathcal{P}, \mathcal{X})$  of  $G$  to a path decomposition  $(\mathcal{P}', \mathcal{X}')$  of  $G'$ , as described in Section 2. By a counting argument using the properties of the nice path decomposition  $(\mathcal{P}, \mathcal{X})$ , we show that the width of  $(\mathcal{P}', \mathcal{X}')$  is at most  $2p' + 1$ , where  $p'$  is the width of  $(\mathcal{P}, \mathcal{X})$ .

## 6.1 Defining an Ordering of Child Blocks

If  $(\mathcal{P}, \mathcal{X})$  is a nice path decomposition of  $G$ , then, for each non-root block  $B$  of  $G$ , at least one bag in  $\mathcal{X}$  contains both the first and last vertices of  $B$  together.

**Definition 3 (Sequence number of a non-root block).** Let  $(\mathcal{P}, \mathcal{X})$  be the nice path decomposition of  $G$  obtained using Lemma 2. For each non-root block  $B$  of  $G$ , we define the sequence number of  $B$  as  $\min\{i \mid X_i \in \mathcal{X} \text{ simultaneously contains both the first and last vertices of } B\}$ .

For each cut vertex  $x$ , there is a unique block  $B^x$  such that  $B^x$  and its child blocks are intersecting at  $x$ . For each cut vertex  $x$ , we define an ordering among the child blocks attached at  $x$ , as follows. If  $B_1, \dots, B_k$  are the child blocks attached at  $x$ , we order them in the increasing order of their sequence numbers in  $(\mathcal{P}, \mathcal{X})$ . If  $B_i$  and  $B_j$  are two child blocks with the same sequence number, their relative ordering is arbitrary.

From the ordering defined, we can make some observations about the appearance of the first and last vertices of a block  $B_i$  in the path decomposition. These observations are crucially used for bounding the width of the path decomposition  $(\mathcal{P}', \mathcal{X}')$  of  $G'$ . Let  $B_1, \dots, B_k$  be the child blocks attached at a cut vertex  $x$ , occurring in that order according to the ordering we defined above. For  $1 \leq i \leq k$ , let  $y_i$  and  $y'_i$  respectively be the last and first vertices of  $B_i$ .

*Property 1.* For any  $1 \leq i \leq k-1$ , if  $\text{Gap}_{\mathcal{X}}(y'_i, y_{i+1}) \neq \emptyset$ , then  $\text{Gap}_{\mathcal{X}}(y'_i, y_{i+1}) = [\text{LastIndex}_{\mathcal{X}}(y'_i) + 1, \text{FirstIndex}_{\mathcal{X}}(y_{i+1})]$  and  $x \in X_t$  for all  $t \in \text{Gap}_{\mathcal{X}}(y'_i, y_{i+1})$ .

*Property 2.* For any  $1 \leq i < j \leq k-1$ ,  $\text{Gap}_{\mathcal{X}}(y'_i, y_{i+1}) \cap \text{Gap}_{\mathcal{X}}(y'_j, y_{j+1}) = \emptyset$ . The proofs of these properties directly follow from the definitions and are given in the full version [1].

## 6.2 Algorithm for Constructing $G'$ and Its Path Decomposition

We use Algorithm 1 to construct  $G'(V, E')$  and a path decomposition  $(\mathcal{P}', \mathcal{X}')$  of  $G'$ . The processing of each cut vertex is done in lines 2 to 7 of Algorithm 1. While processing a cut vertex  $x$ , the algorithm adds the edges  $(y'_1, y_2), (y'_2, y_3), \dots, (y'_{k_x-1}, y_{k_x})$  (as defined in the algorithm) and modifies the path decomposition, to reflect each edge addition.

**Lemma 5.**  $G'$  is outerplanar and for each cut vertex  $x$  of  $G'$ ,  $G' \setminus x$  has exactly two components.

We can prove this by applying Lemma 3, following the addition of each edge in  $E' \setminus E$  by Algorithm 1. Refer to the full version [1] for the proof.

**Lemma 6.**  $(\mathcal{P}', \mathcal{X}')$  is a path decomposition of  $G'$  of width at most  $8p + 7$ .

*Proof.* Algorithm 1 initialized  $(\mathcal{P}', \mathcal{X}')$  to  $(\mathcal{P}, \mathcal{X})$  and modified it during each edge addition. By Property 1, we have  $\text{Gap}_{\mathcal{X}}(y'_i, y_{i+1}) = [\text{LastIndex}_{\mathcal{X}}(y'_i) + 1, \text{FirstIndex}_{\mathcal{X}}(y_{i+1})]$ . Hence, by the modification done in lines 7 to 7 while adding a new edge  $(y'_i, y_{i+1})$ ,  $(\mathcal{P}', \mathcal{X}')$  becomes a path decomposition of the

**Algorithm 1.** Computing  $G'$  and its path decomposition

---

**Input:** An outerplanar graph  $G(V, E)$  and a nice path decomposition  $(\mathcal{P}, \mathcal{X})$  of  $G$ , the rooted block tree of  $G$ , the Hamiltonian cycle of each non-trivial block of  $G$  and the first and last vertices of each non-root block of  $G$

**Output:** An outerplanar supergraph  $G'(V, E')$  of  $G$  such that, for every cut vertex  $x$  of  $G'$ ,  $G' \setminus x$  has exactly two connected components, a path decomposition  $(\mathcal{P}', \mathcal{X}')$  of  $G'$

```

1  $E' = E$ ,  $(\mathcal{P}', \mathcal{X}') = (\mathcal{P}, \mathcal{X})$ 
2 for each cut vertex  $x \in V(G)$  do
3   Let  $B_1, \dots, B_{k_x}$ , in that order, be the child blocks attached at  $x$ , according
      to the ordering defined in Section 6.1
4   for  $i = 1$  to  $k_x - 1$  do
5     Let  $y'_i$  be the first vertex of  $B_i$  and  $y_{i+1}$  be the last vertex of  $B_{i+1}$ 
6      $E' = E' \cup \{(y'_i, y_{i+1})\}$ 
7     if  $\text{Gap}_{\mathcal{X}}(y'_i, y_{i+1}) \neq \emptyset$  then for  $t \in \text{Gap}_{\mathcal{X}}(y'_i, y_{i+1})$  do  $X'_t = X'_t \cup \{y'_i\}$ 

```

---

graph containing the edge  $(y'_i, y_{i+1})$ , by the method explained in Section 2. It follows that  $(\mathcal{P}', \mathcal{X}')$  is a path decomposition of  $G'$ .

Consider any  $X'_t \in \mathcal{X}'$ . While processing the cut vertex  $x$ , if Algorithm 1 inserts a new vertex  $y'_i$  to  $X'_t$ , to reflect the addition of a new edge  $(y'_i, y_{i+1})$  then,  $t \in \text{Gap}_{\mathcal{X}}(y'_i, y_{i+1})$ . Suppose  $(y'_i, y_{i+1})$  and  $(y'_j, y_{j+1})$  are two new edges added while processing the cut vertex  $x$ , where,  $1 \leq i < j \leq k_x - 1$ . By property 2, we know that if  $t \in \text{Gap}_{\mathcal{X}}(y'_i, y_{i+1})$ , then,  $t \notin \text{Gap}_{\mathcal{X}}(y'_j, y_{j+1})$ . Therefore, when the algorithm is processing a cut vertex  $x$  in lines 2 to 7, at most one vertex is newly getting inserted to  $X'_t$ . Moreover, if  $t \in \text{Gap}_{\mathcal{X}}(y'_i, y_{i+1})$  then, the cut vertex  $x \in X_t$ , by Property 1. It follows that  $|X'_t| \leq |X_t| + \text{number of cut vertices in } X_t \leq 2|X_t| \leq 2(4p + 4)$ . Therefore, the width of the path decomposition  $(\mathcal{P}', \mathcal{X}')$  is at most  $8p + 7$ .  $\square$

## 7 Construction of $G''$ and Its Path Decomposition

In this section, we give an algorithm to add some more edges to  $G'(V, E')$  so that the resultant graph  $G'''(V, E'')$  is bi-connected. The algorithm also extend the path decomposition  $(\mathcal{P}', \mathcal{X}')$  of  $G'$  to a path decomposition  $(\mathcal{P}'', \mathcal{X}'')$  of  $G''$ . An analysis of the algorithm shows the existence of a surjective mapping from the cut vertices of  $G'$  to the edges in  $E'' \setminus E'$ . A counting argument based on the surjective mapping shows that the width of the path decomposition  $(\mathcal{P}'', \mathcal{X}'')$  is at most  $16p + 15$ . For making our presentation simpler, if a block  $B_i$  is just an edge  $(u, v)$ , we abuse the definition of a Hamiltonian cycle and say that  $u$  and  $v$  are clockwise neighbors of each other in the Hamiltonian cycle of  $B_i$ .

Recall that the graph  $G'$  has the property that for every cut vertex  $x$  of  $G'$ ,  $G' \setminus x$  has exactly two components. Since any cut vertex belongs to exactly two blocks of  $G$ , based on the rooted block tree structure of  $G$ , we call them as the parent block containing  $x$  and the child block containing  $x$ . We use  $\text{child}_x(B)$  to denote the child block of the block  $B$  at the cut vertex  $x$  and  $\text{parent}(B)$  to

denote the parent block of the block  $B$ . For a block  $B$ ,  $\text{next}_B(v)$  denotes the successor of the vertex  $v$  in the Hamiltonian cycle of  $B$ .

To get an intuition about our algorithm, the reader may consider it as a traversal of vertices of  $G'$ , starting from a non-cut vertex in the root block of  $G'$  and proceeding to the successor of  $v$  on reaching a non-cut vertex  $v$ . On reaching a cut vertex  $x$ , the algorithm recursively traverses the child block containing  $x$  and its descendant blocks and comes back to  $x$  to continue the traversal of the remaining graph. However, before starting the recursive traversal of the child block containing  $x$  and its descendant blocks, the algorithm sets  $\text{bypass}(x) = \text{TRUE}$ . (Note that, since there is only one child block attached to any cut vertex, each cut vertex is bypassed only once.) In this way, when a sequence of one or more cut vertices is bypassed, an edge is added from the vertex preceding the first bypassed vertex in the sequence to the vertex succeeding the last bypassed vertex in the sequence. The path decomposition is also modified, to reflect this edge addition. The detailed algorithm to bi-connect  $G'$  is given in Algorithm 2. The following Lemma summarizes some observations about how Algorithm 2 works. A proof this lemma can be found in the full version [1].

---

**Algorithm 2.** Computing a bi-connected outerplanar supergraph

---

**Input:** An outerplanar graph  $G'(V, E')$  such that  $G' \setminus x$  has exactly two connected components for every cut vertex  $x$  of  $G'$ . A path decomposition  $(\mathcal{P}', \mathcal{X}')$  of  $G'$ . The rooted block tree of  $G'$ , the Hamiltonian cycle of each non-trivial block of  $G'$  and the first and last vertices of each non-root block of  $G'$

**Output:** A bi-connected outerplanar supergraph  $G''(V, E'')$  of  $G'$ , a path decomposition  $(\mathcal{P}'', \mathcal{X}'')$  of  $G''$

- 1  $E'' = E'$ ,  $(\mathcal{P}'', \mathcal{X}'') = (\mathcal{P}', \mathcal{X}')$
- 2 **for** each vertex  $v \in V(G')$  **do**
- 3      $\text{completed}(v) = \text{FALSE}$ , **if**  $v$  is a cut vertex **then**  $\text{bypass}(v) = \text{FALSE}$
- 4     Choose  $v$  to be some non-cut vertex of the root block
- 5      $B = \text{root block}$ ,  $\text{completed}(v) = \text{TRUE}$ ,  $\text{completedCount} = 1$
- 6 **while**  $\text{completedCount} < |V(G')|$  **do**
- 7      $v' = \text{next}_B(v)$
- 8     **while**  $v'$  is a cut vertex and  $\text{bypass}(v')$  is FALSE **do**
- 9          $\text{bypass}(v') = \text{TRUE}$ ,  $B = \text{child}_{v'}(B)$ ,  $v' = \text{next}_B(v')$
- 10      **if**  $v'$  is a cut vertex and  $\text{bypass}(v')$  is TRUE **then**  $B = \text{parent}(B)$
- 11       $\text{completed}(v') = \text{TRUE}$ ,  $\text{completedCount} = \text{completedCount} + 1$
- 12      **if**  $(v, v')$  is not an edge in  $G'$  **then**
- 13          $E'' = E'' \cup \{(v, v')\}$
- 14      **if**  $\text{Gap}_{\mathcal{X}'}(v, v') \neq \emptyset$  **then**
- 15         **if**  $\text{LastIndex}_{\mathcal{X}'}(v) < \text{FirstIndex}_{\mathcal{X}'}(v')$  **then** **for**  $t \in \text{Gap}_{\mathcal{X}'}(v, v')$
- 16             **do**  $X''_t = X''_t \cup \{v\}$
- 17         **else if**  $\text{LastIndex}_{\mathcal{X}'}(v') < \text{FirstIndex}_{\mathcal{X}'}(v)$  **then** **for**  $t \in \text{Gap}_{\mathcal{X}'}(v, v')$  **do**  $X''_t = X''_t \cup \{v'\}$
- 18      $v = v'$

---

- Lemma 7.** 1. Inside a block, the algorithm traverses vertices in the clockwise order of the unique Hamiltonian cycle of the block.  
 2. When the algorithm encounters a non-cut vertex  $x$  during the traversal, it declares that  $x$  is completed.  
 3. The algorithm encounters a cut vertex  $x$  for the first time, while traversing the parent block containing  $x$ . Then, the algorithm bypasses  $x$  (i.e. set  $\text{bypass}(x) = \text{TRUE}$ ) and descends to the child block containing  $x$  and start traversing the child block from the successor of  $x$  in the child block's Hamiltonian cycle.  
 4. When the algorithm encounters a cut vertex  $x$  for a second time, the current block being traversed is the child block containing  $x$ . Then the algorithm traverses  $x$  and declare that  $x$  is completed and ascends to the parent block containing  $x$ . Then it continues the traversal of the parent block containing  $x$ , by considering the successor of  $x$  in the parent block's Hamiltonian cycle.  
 5. When the algorithm declares that a cut vertex  $x$  is completed, all vertices of the child block containing  $x$  and all its descendant blocks have been completed.  
 6. Every vertex is encountered at least once. Every vertex is completed and a vertex which is declared completed is never encountered again. When  $\text{completedCount} = |V(G')|$ , all the vertices of the graph have been completed.  
 7. When the algorithm is bypassing a sequence of one or more cut vertices, an edge is added from the vertex preceding the first bypassed vertex in the sequence to the vertex succeeding the last bypassed vertex in the sequence and the path decomposition is modified, to reflect this edge addition.  
 8. Every new edge added has a sequence of bypassed cut vertices associated with it. If  $x_1, x_2, \dots, x_k$  is the sequence of bypassed cut vertices associated with an edge  $(u, v) \in E'' \setminus E'$ , then  $u, x_1, x_2, \dots, x_k, v$  is a path in  $G'$ . Each cut vertex of  $G'$  is bypassed exactly once in our traversal and hence associated with a unique edge in  $E'' \setminus E'$ .

**Lemma 8.**  $G''$  is bi-connected.

*Proof.* We show that  $G''$  does not have any cut vertices. Since  $G''$  is a supergraph of  $G'$ , if a vertex  $x$  is not a cut vertex in  $G'$ , it will not be a cut vertex in  $G''$ . We need to show that the cut vertices in  $G'$  become non-cut vertices in  $G''$ . Consider a newly added edge  $(u, v)$  of  $G''$ . Without loss of generality, assume that  $u$  was completed before  $v$  in the traversal, and  $(x_1, x_2, \dots, x_k)$  is the sequence of bypassed cut vertices associated with the edge  $(u, v)$ . When our algorithm adds the edge  $(u, v)$ , it creates the cycle  $u, x_1, x_2, \dots, x_k, v, u$  in the resultant graph. Recall that, for each  $1 \leq i \leq k$ ,  $G' \setminus x_i$  had exactly two components; one containing  $x_{i-1}$  and the other containing  $x_{i+1}$ . After the addition of the edge, vertices  $x_{i-1}$ ,  $x_i$  and  $x_{i+1}$  lie on a common cycle. Hence, when the edge  $(u, v)$  is added, for  $1 \leq i \leq k$ ,  $x_i$  is no longer a cut vertex. Since every cut vertex in  $G'$  was part of the bypass sequence associated with some edge in  $E'' \setminus E'$ , all of them become non-cut vertices in  $G''$ .  $\square$

**Lemma 9.**  $G''$  is outerplanar.

For a proof of this lemma, refer to the full version [1].

**Lemma 10.**  $(\mathcal{P}'', \mathcal{X}'')$  is a path decomposition of  $G''$  of width at most  $16p+15$ .

*Proof.* It is clear that  $(\mathcal{P}'', \mathcal{X}'')$  is a path decomposition of  $G''$ , since we constructed it using the method explained in Section 2.

For each  $1 \leq i \leq m$ , let  $S_i = \{x_1, \dots, x_k\}$  denote the set of cut vertices that belong to the bypassed cut vertex sequence associated with the edge  $e_i = (u_i, v_i) \in E'' \setminus E'$ . While adding the edge  $e_i$ , a vertex was inserted into  $X_t'' \in \mathcal{X}''$  only if  $t \in \text{Gap}_{\mathcal{X}'}(u_i, v_i)$ . We will now show that, if  $t \in \text{Gap}_{\mathcal{X}'}(u_i, v_i)$ , then,  $X_t' \cap S_i \neq \emptyset$ . Without loss of generality, assume that  $\text{LastIndex}_{\mathcal{X}'}(u_i) < \text{FirstIndex}_{\mathcal{X}'}(v_i)$ . Let  $x_1, \dots, x_k$  be the sequence of cut vertices bypassed while adding the edge  $(u_i, v_i)$ . Since  $u_i$  is adjacent to  $x_1$ , both of them are together present in some bag in  $X_t' \in \mathcal{X}'$ , with  $t \leq \text{LastIndex}_{\mathcal{X}'}(u_i)$ . Similarly, since  $v_i$  is adjacent to  $x_k$ , they both are together present in some bag  $X_t' \in \mathcal{X}'$ , with  $t \geq \text{FirstIndex}_{\mathcal{X}'}(v_i)$ . The sequence  $x_1, \dots, x_k$  is a path in  $G'$  between  $x_1$  and  $x_k$ . Therefore, every bag in  $X_t' \in \mathcal{X}'$  with  $t \in \text{Gap}_{\mathcal{X}'}(u_i, v_i)$  should contain at least one of the cut vertices from the set  $S_i$ .

Thus, by the modification done to the path decomposition to reflect the addition of the edge  $e_i$ , the size of each bag in  $X_t'' \in \mathcal{X}''$  with  $t \in \text{Gap}_{\mathcal{X}'}(u_i, v_i)$  increases by exactly one and in that case,  $X_t' \cap S_i \neq \emptyset$ . The other bags are unaffected by this modification. Therefore, for any  $t$  in the index set,  $|X_t''| = |X_t'| + |\{i \mid 1 \leq i \leq m, S_i \cap X_t' \neq \emptyset\}|$ . But,  $|\{i \mid 1 \leq i \leq m, S_i \cap X_t' \neq \emptyset\}| \leq |X_t'|$ , because  $S_i \cap S_j = \emptyset$ , for  $1 \leq i < j \leq m$ , by part 8 of Lemma 7. Therefore, for any  $t$ ,  $|X_t''| \leq 2|X_t'| \leq 2(8p+8)$ . Therefore, width of the path decomposition  $(\mathcal{P}'', \mathcal{X}'')$  is at most  $16p+15$ .  $\square$

## 8 Efficiency

The preprocessing step of computing a rooted block tree of the given outerplanar graph  $G$  and finding the Hamiltonian cycles of each non-trivial block can be done in linear time [4,8,14]. The special tree decomposition in Govindan et al.[7] is also computable in linear time. Using the Hamiltonian cycle of each non-trivial block, we did only a linear time modification in Section 4, to produce the nice tree decomposition  $(T, \mathcal{Y})$  of  $G$ . An optimal path decomposition of the tree  $T$ , of total size  $O(n \text{ pw}(T))$  can be computed in time  $O(n \text{ pw}(T))$ [12]. The time taken is  $O(n \log n)$ , since outerplanar graphs have pathwidth at most  $\log n$ , and  $T$  was a spanning tree of the outerplanar graph  $G$ . For computing the nice path decomposition  $(\mathcal{P}, \mathcal{X})$  of  $G$  in Section 4, the time spent is linear in the size of the path decomposition obtained for  $T$ , i.e,  $O(n \log n)$  and the total size of  $(\mathcal{P}, \mathcal{X})$  is  $O(n \log n)$ . Computing the FirstIndex, LastIndex and Range of vertices and the sequence number of blocks can be done in time linear in the size of the path decomposition. Since the resultant graph is outerplanar, Algorithm 1 and Algorithm 2 adds only a linear number of new edges. Since the size of each bag in the path decompositions  $(\mathcal{P}', \mathcal{X}')$  of  $G'$  and  $(\mathcal{P}'', \mathcal{X}'')$  of  $G''$  are only a constant times the size of the corresponding bag in  $(\mathcal{P}, \mathcal{X})$ , the time taken for modifying  $(\mathcal{P}, \mathcal{X})$  to obtain  $(\mathcal{P}', \mathcal{X}')$  and later modifying it to  $(\mathcal{P}'', \mathcal{X}'')$  takes

only time linear in size of  $(\mathcal{P}, \mathcal{X})$ ; i.e.,  $O(n \log n)$  time. Hence, the time spent in constructing  $G''$  and its path decomposition of width  $O(\text{pw}(G))$  is  $O(n \log n)$ .

## 9 Conclusion

In this paper, we have described a  $O(n \log n)$  time algorithm to add edges to a given outerplanar graph  $G$  of pathwidth  $p$  to get a bi-connected outerplanar graph  $G''$  of pathwidth at most  $16p + 15$ . We also get the corresponding path decomposition of  $G''$  in  $O(n \log n)$  time. Our technique is to produce a nice path decomposition of  $G$  and make use of the properties of this decomposition, while adding the new edges. Our algorithm can be used as a preprocessing step, in the algorithm proposed by Biedl [3], to produce a planar drawing of  $G$  on a grid of height  $O(p)$ . As explained by Biedl [3], this is a constant factor approximation algorithm, to get a planar drawing of  $G$  of minimum height.

## References

1. Babu, J., Basavaraju, M., Chandran, L.S., Rajendraprasad, D.: 2-connecting outerplanar graphs without blowing up the pathwidth. CoRR abs/1212.6382 (2012), <http://arxiv.org/abs/1212.6382>
2. Biedl, T.: Small drawings of outerplanar graphs, series-parallel graphs, and other planar graphs. *Discrete Comput. Geom.* 45(1), 141–160 (2011)
3. Biedl, T.: A 4-approximation for the height of 2-connected outer-planar graph drawings. In: WAOA 2012 (2012)
4. Chartrand, G., Harary, F.: Planar permutation graphs. *Annales de l'institut Henri Poincaré (B) Probabilités et Statistiques* 3, 433–438 (1967)
5. Dujmovic, V., Morin, P., Wood, D.R.: Path-width and three-dimensional straight-line grid drawings of graphs. In: Goodrich, M.T., Kobourov, S.G. (eds.) *GD 2002. LNCS*, vol. 2528, pp. 42–53. Springer, Heidelberg (2002)
6. García, A., Hurtado, F., Noy, M., Tejel, J.: Augmenting the connectivity of outerplanar graphs. *Algorithmica* 56(2), 160–179 (2010)
7. Govindan, R., Langston, M.A., Yan, X.: Approximating the pathwidth of outerplanar graphs. *Inf. Process. Lett.* 68(1), 17–23 (1998)
8. Hopcroft, J., Tarjan, R.: Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM* 16(6), 372–378 (1973)
9. Kant, G.: Augmenting outerplanar graphs. *Journal of Algorithms* 21(1), 1–25 (1996)
10. Robertson, N., Seymour, P.D.: Graph minors. iii. planar tree-width. *J. Comb. Theory, Ser. B* 36(1), 49–64 (1984)
11. Schnyder, W.: Embedding planar graphs on the grid. In: Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1990, pp. 138–148 (1990)
12. Skodinis, K.: Construction of linear tree-layouts which are optimal with respect to vertex separation in linear time. *J. Algorithms* 47(1), 40–59 (2003)
13. Suderman, M.: Pathwidth and layered drawings of trees. *Int. J. Comput. Geometry Appl.* 14(3), 203–225 (2004)
14. Syslo, M.M.: Characterizations of outerplanar graphs. *Discrete Mathematics* 26(1), 47–53 (1979)

# How to Catch $L_2$ -Heavy-Hitters on Sliding Windows\*

Vladimir Braverman<sup>1,\*\*\*</sup>, Ran Gelles<sup>2</sup>, and Rafail Ostrovsky<sup>2,3,\*\*\*</sup>

<sup>1</sup> Department of Computer Science, Johns Hopkins University  
[vova@cs.jhu.edu](mailto:vova@cs.jhu.edu)

<sup>2</sup> Department of Computer Science, University of California, Los Angeles  
[gelles@cs.ucla.edu](mailto:gelles@cs.ucla.edu)

<sup>3</sup> Department of Mathematics, University of California, Los Angeles  
[rafael@cs.ucla.edu](mailto:rafael@cs.ucla.edu)

**Abstract.** Finding heavy-elements (heavy-hitters) in streaming data is one of the central, and well-understood tasks. Despite the importance of this problem, when considering the *sliding windows* model of streaming (where elements eventually expire) the problem of finding  $L_2$ -heavy elements has remained completely open despite multiple papers and considerable success in finding  $L_1$ -heavy elements.

Since the  $L_2$ -heavy element problem doesn't satisfy certain conditions, existing methods for sliding windows algorithms, such as smooth histograms or exponential histograms are not directly applicable to it. In this paper, we develop the first polylogarithmic-memory algorithm for finding  $L_2$ -heavy elements in the sliding window model.

Our technique allows us not only to find  $L_2$ -heavy elements, but also heavy elements with respect to any  $L_p$  with  $0 < p \leq 2$  on sliding windows. By this we completely “close the gap” and resolve the question of finding  $L_p$ -heavy elements in the sliding window model with polylogarithmic memory, since it is well known that for  $p > 2$  this task is impossible.

We demonstrate a broader applicability of our method on two additional examples: we show how to obtain a sliding window approximation of the similarity of two streams, and of the fraction of elements that appear exactly a specified number of times within the window (the  $\alpha$ -rarity problem). In these two illustrative examples of our method, we replace the current *expected* memory bounds with *worst case* bounds.

---

\* A preliminary full version of this paper appears online [10].

\*\* This work was supported in part by DARPA grant N660001-1-2-4014.

\*\*\* Research supported in part by NSF grants CNS-0830803; CCF-0916574; IIS-1065276; CCF-1016540; CNS-1118126; CNS-1136174; US-Israel BSF grant 2008411, OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Foundation Award, Teradata Research Award, and Lockheed-Martin Corporation Research Award. This material is also based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0392. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

## 1 Introduction

A *data stream*  $S$  is an ordered multiset of elements  $\{a_0, a_1, a_2 \dots\}$  where each element  $a_t \in \{1, \dots, u\}$  arrives at time  $t$ . In the *sliding window model* we consider at each time  $t \geq N$  the last  $N$  elements of the stream, i.e. the window  $W = \{a_{t-(N-1)}, \dots, a_t\}$ . These elements are called *active*, whereas elements that arrived prior to the current window  $\{a_i \mid 0 \leq i < t - (N - 1)\}$  are *expired*. For  $t < N$ , the window consists of all the elements received so far,  $\{a_0, \dots, a_t\}$ .

Usually, both  $u$  and  $N$  are considered to be extremely large so it is not applicable to save the entire stream (or even one entire window) in memory. The problem is to be able to calculate various characteristics about the window's elements using small amount of memory (usually, polylogarithmic in  $N$  and  $u$ ). We refer the reader to the books of Muthukrishnan [40] and Aggarwal (ed.) [1] for extensive surveys on data stream models and algorithms.

One of the main open problems in data streams deals with the relations between the different streaming models [38], specifically between the unbounded stream model and the sliding window model. In this paper we provide another important step in clarifying the connection between these two models by showing that finding  $L_p$ -heavy hitters is just as doable on sliding windows as on the entire stream.

We focus on approximation-algorithms for certain statistical characteristics of the data streams, specifically, finding frequent elements. The problem of finding frequent elements in a stream is useful for many applications, such as network monitoring [43] and DoS prevention [24,19,4], and was extensively explored over the last decade (see [40,18] for a definition of the problem and a survey of existing solutions, as well as [14,37,27,33,17,3,20,45,28]).

We say that an element is *heavy* if it appears more times than a constant fraction of some  $L_p$  norm of the stream. Recall that for  $p > 0$ , the  $L_p$  norm of the frequency vector<sup>1</sup> is defined by  $L_p = (\sum_i n_i^p)^{1/p}$ , where  $n_i$  is the frequency of element  $i \in [u]$ , i.e., the number of times  $i$  appears in the window. Since different  $L_p$  can be considered, we obtain several different ways to define a “heavy” element. Generally speaking (as mentioned in [31]), when considering frequent elements (heavy-hitters) with respect to  $L_p$ , the higher  $p$  is, the better. Specifically, identifying frequent elements with respect to  $L_2$  is better than  $L_1$  since an  $L_1$  algorithm can always be replaced with an  $L_2$  algorithm, with less or equal memory consumption (but not vice versa).

Naturally, finding frequent elements with respect to the  $L_2$  norm is a more difficult task (memory-wise) than the equivalent  $L_1$  problem. To demonstrate this fact let us regard the following example: let  $S$  be a stream of size  $N$ , in which the element  $a_1$  appears  $\sqrt{N}$  times, while the rest of the elements  $a_2, \dots, a_{N-\sqrt{N}}$  appear exactly once in  $S$ . Say we wish to identify  $a_1$  as an heavy element. Note that  $n_1 = \frac{1}{\sqrt{N}}L_1$  while  $n_1 = cL_2$ , where  $c$  is a constant, lower bounded by  $\frac{1}{\sqrt{2}}$ . Therefore, as  $N$  grows,  $n_1/L_1 \rightarrow 0$  goes to zero, while  $n_1/L_2$  is bounded

---

<sup>1</sup> Throughout the paper we use the term “ $L_p$  norm” to indicate the  $L_p$  norm of the frequency vector, i.e., the  $p$ th root of the  $p$ th frequency moment  $F_p = \sum_i n_i^p$  [2], rather than the norm of the data itself.

by a constant. If an algorithm finds elements which are heavier than  $\gamma L_p$  with memory  $\text{poly}(\gamma^{-1}, \log N, \log u)$ , then for  $p = 2$  we get a polylogarithmic memory, while for  $p = 1$  the memory consumption is super-logarithmic.

We focus on solving the following  $L_2$ -heaviness problem:

**Definition 1.1 (( $\gamma, \epsilon$ )-approximation of  $L_2$ -frequent elements).** For  $0 < \epsilon, \gamma < 1$ , output any element  $i \in [u]$  such that  $n_i > \gamma L_2$  and no element such that  $n_i < (1 - \epsilon)\gamma L_2$ .

The  $L_2$  norm is the most powerful norm for which we can expect a polylogarithmic solution, for the frequent-elements problem. This is due to the known lower bound of  $\Omega(u^{1-2/p})$  for calculating  $L_p$  over a stream [42,6].

There has been a lot of progress on the question of finding  $L_1$ -frequent elements, in the sliding window model [3,45,28], however those algorithms cannot be used to find  $L_2$ -frequent elements with an efficient memory. In 2002, Charikar, Chen and Farach-Colton [14] developed the COUNTSKETCH algorithm that can approximate the “top  $k$ ” frequent-elements on *an unbounded stream*, where  $k$  is given as an input. Formally, their algorithm outputs only elements with frequency larger than  $(1 - \epsilon)\phi_k$ , where  $\phi_k$  is the frequency of the  $k$ th most frequent element in the stream, using memory proportional to  $L_2^2/(\epsilon\phi_k)^2$ . Since the “heaviness” in this case is relative to  $\phi_k$ , and the memory is bounded by the fraction  $L_2^2/(\epsilon\phi_k)^2$ , Charikar et al.’s algorithm finds in fact heaviness in terms of the  $L_2$  norm. A natural question is whether one can develop an algorithm for finding frequent-elements that appear at least  $\gamma L_2$  times in the *sliding window model*, using  $\text{poly}(\gamma^{-1}, \log N, \log u)$  memory.

**Our Results.** We give the first polylogarithmic algorithm for finding an  $\epsilon$ -approximation of the  $L_2$ -frequent elements in the sliding window model. Our algorithm is able to identify elements that appear within the window a number of times which is at least a  $\gamma$ -fraction of the  $L_2$  norm of the window, up to a multiplicative factor of  $(1 - \epsilon)$ . In addition, the algorithm guarantees to output *all* the elements with frequency at least  $(1 + \epsilon)\gamma L_2$ .

**Theorem 1.2.** *There exists an efficient sliding window algorithm that outputs a  $(\gamma, \epsilon)$ -approximation of the  $L_2$ -frequent-elements, with probability at least  $1 - \delta$  and memory  $\text{poly}(\epsilon^{-1}, \gamma^{-1}, \log N, \log \delta^{-1})$ .*

We note that the COUNTSKETCH algorithm works in the unbounded model and does not apply directly on sliding windows. Moreover, COUNTSKETCH solves a slightly different (yet related) problem, namely, the top- $k$  problem, rather than the  $L_2$  heaviness. To achieve our result on  $L_2$  heavy hitters, we combine in a non-trivial way the scheme of Charikar et al. with a sliding-window approximation for  $L_2$  as given by Braverman and Ostrovsky [9]. Variants of these techniques sufficient to derive similar results were known since 2002,<sup>2</sup> however no algorithm for  $L_2$  heavy hitters was reported despite several papers on  $L_1$  heavy hitters.

---

<sup>2</sup> Indeed, we use the algorithm of Charikar et al. [14] that is known since 2002. Also, it is possible to replace (with some non-trivial effort) our smooth histogram method for  $L_2$  computation with the algorithm of Datar, Gionis, Indyk and Motwani [22] for  $L_2$  approximation.

Our solution gives another step in the direction of making a connection between the unbounded and sliding window models, as it provides an answer for the very important question of heavy hitters in the sliding window model. The result joins the various solutions of finding  $L_1$ -heavy hitters in sliding windows [27,3,41,4,45,28,29], and can be used in various algorithms that require identifying  $L_2$  heavy hitters, such as [32,8] and others. More generally, our paper resolves the question of finding  $L_p$ -heavy elements on sliding windows for all values of  $p$  that allows small memory one-pass solutions (i.e. for  $0 < p \leq 2$ ). By this we completely close the gap between the case of  $p \leq 1$ , solved by previous works, and the impossibility result for the case of  $p > 2$ .

**A Broader Perspective.** In fact, one can consider the tools we develop for the frequent elements problem as a general method that allows obtaining a sliding window solution out of an algorithm for the unbounded model, for a wide range of functions. We introduce a new concept which uses a smooth-histogram in order to perform sliding window approximation of *non-smooth* properties. Informally speaking, the main idea is to relate the non-smooth property  $f$  with some other, smooth<sup>3</sup>, property  $g$ , such that changes in  $f$  are bounded by the changes in  $g$ . By maintaining a smooth-histogram for the smooth function  $g$ , we *partition* the stream into sets of sub-streams (buckets). Due to the properties of the smooth-histogram we can bound the error (of approximating  $g$ ) for every sub-stream, and thus get an approximation of  $f$ . We use the term *semi-smooth* to describe these kinds of algorithms.

We demonstrate the above idea by showing a concrete efficient sliding window algorithm for the properties of rarity and similarity [21]; we stress that neither is smooth (see Section 4 for definitions of these problems). In addition to the properties of rarity and similarity, we believe that the tools we develop here can be used to build efficient sliding window approximations for many other (non-smooth) properties and provide a general new method for computing on sliding windows. Indeed, in a subsequent work Tirthapura and Woodruff [44] use our methods to compute various *correlated aggregations*. It is important to note that trying to build a smooth-histogram (or any other known sketch) directly to  $f$  will not preserve the required invariants, and the memory consumption might not be efficient.

**Previous Works.** *Frequent Element Problem.* Finding elements that appear many times in the stream (“heavy hitters”) is a very central question and thus has been extensively studied both for the unbounded model [23,35,17,39] and for the sliding window model [3,41,45,28] as well as other variants such as the *offline stream* model [37], insertion and deletion model [20,33], finding heavy-distinct-hitter [4], etc. Reducing the processing time was done by [36] into  $O(\frac{1}{\epsilon})$  and by [29] into  $O(1)$ .

Another problem which is related to finding the heavy hitters, is the top- $k$  problem, namely, finding the  $k$  most frequent elements. As mentioned above, Charikar, Chen and Farach-Colton [14] provide an algorithm that finds the  $k$

---

<sup>3</sup> Of course, other kinds of aggregations can be used, however our focus is on smooth histograms.

most frequent elements in the unbounded model (up to a precision of  $1 \pm \epsilon$ ). Golab, DeHaan, Demaine, López-Ortiz and Munro [27] solve this problem in the *jumping window* model.

*Similarity and  $\alpha$ -Rarity Problem.* The similarity problem was defined in order to give a rough estimation of closeness between files over the web [12] (and independently in [15]). Later, it was shown how to use min-hash functions [30] in order to sample from the stream, and estimate the similarity of two streams.

The notion of  $\alpha$ -rarity, introduced by Datar and Muthukrishnan [21], is that of finding the fraction of elements that appear exactly  $\alpha$  times within the stream. This quantity can be seen as finding the fraction of elements with frequency within certain bounds.

The questions of rarity and similarity were analyzed, both for the unbounded stream and the sliding window models, by Datar and Muthukrishnan [21], achieving an expected memory bound of  $O(\log N + \log u)$  words of space for constant  $\epsilon, \alpha, \delta$ . At the bit level, their algorithm requires  $O(\alpha \cdot \epsilon^{-3} \log \delta^{-1} \log N (\log N + \log u))$  bits for  $\alpha$ -rarity and  $O(\epsilon^{-3} \log \delta^{-1} \log N (\log N + \log u))$  bits for similarity, with  $1 - \delta$  being the probability of success<sup>4</sup>. Our techniques improve these results and obtain a *worst case* memory consumption of essentially the same magnitude (up to a factor of  $\log \log N$ ).

## 2 Preliminaries

*Notations.* We say that an algorithm  $A_f$  is an  $(\epsilon, \delta)$ -approximation of a function  $f$ , if for any input  $S$ ,  $(1 - \epsilon)f(S) \leq A_f(S) \leq (1 + \epsilon)f(S)$ , except with probability  $\delta$  over  $A_f$ 's coin tosses. We denote this relation as  $A_f \in (1 \pm \epsilon)f$  for short. We denote an output of an approximation algorithm with a hat symbol, e.g., the estimator of  $f$  is denoted  $\hat{f}$ .

The set  $\{1, 2, \dots, n\}$  is usually denoted by  $[n]$ . If a stream  $B$  is a suffix of  $A$ , we denote  $B \subseteq_r A$ . For instance, let  $A = \{q_1, q_2, \dots, q_n\}$  then  $B = \{q_{n_1}, q_{n_1+1}, \dots, q_n\} \subseteq_r A$ , for  $1 \leq n_1 \leq n$ . The notation  $A \cup C$  denotes the concatenation of the stream  $C = \{c_1, c_2, \dots, c_m\}$  to the end of stream  $A$ , i.e.,  $A \cup C = \{q_1, q_2, \dots, q_n, c_1, c_2, \dots, c_m\}$ . The notation  $|A|$  denotes the number of different elements in the stream  $A$ , that is the cardinality of the *set* induced by the multiset  $A$ . The size of the stream (i.e. of the multiset)  $A$  will be denoted as  $\|A\|$ , e.g., for the example above  $\|A\| = n$ .

We use the notation  $\tilde{O}(\cdot)$  to indicate an asymptotic bound which suppresses terms of magnitude  $\text{poly}(\log \frac{1}{\epsilon}, \log \log \frac{1}{\delta}, \log \log N, \log \log u)$ .

Due to a strict page limit, we defer all proofs to the full version [10].

*Smooth Histograms.* Recently, Braverman and Ostrovsky [9] showed that a function  $f$  can be  $\epsilon$ -approximated in the sliding window model, if  $f$  is a *smooth function*, and if it can be calculated (or approximated) in the unbounded stream model. Formally,

---

<sup>4</sup> These bounds are not explicitly stated in [21], but follow from the analysis (see Lemma 1 and Lemma 2 in [21]).

**Definition 2.1.** A polynomial function  $f$  is  $(\alpha, \beta)$ -smooth if it satisfies the following properties: (i)  $f(A) \geq 0$ ; (ii)  $f(A) \geq f(B)$  for  $B \subseteq_r A$ ; and (iii) there exist  $0 < \beta \leq \alpha < 1$  such that if  $(1 - \beta)f(A) \leq f(B)$  for  $B \subseteq_r A$ , then  $(1 - \alpha)f(A \cup C) \leq f(B \cup C)$  for any  $C$ .

If an  $(\alpha, \beta)$ -smooth  $f$  can be calculated (or  $(\epsilon, \delta)$ -approximated) on an unbounded stream with memory  $g(\epsilon, \delta)$ , then there exists an  $(\alpha + \epsilon, \delta)$ -estimation of  $f$  in the sliding window model using  $O(\frac{1}{\beta} \log N(g(\epsilon, \frac{\delta\beta}{\log N}) + \log N))$  bits [9].

The key idea is to construct a “smooth-histogram”, a structure that contains estimations on  $O(\frac{1}{\beta} \log N)$ -suffixes of the stream,  $A_1 \supseteq_r A_2 \supseteq_r \dots \supseteq_r A_{c \frac{1}{\beta} \log(n)}$ . Each suffix  $A_i$  is called a *Bucket*. Each new element in the stream initiates a new bucket, however adjacent buckets with a close estimation value are removed (keeping only one representative). Since the function is “smooth”, i.e., monotonic and slowly-changing, it is enough to save  $O(\frac{1}{\beta} \log N)$  buckets in order to maintain a reasonable approximation of the window. At any given time, the current window  $W$  is between buckets  $A_1$  and  $A_2$ , i.e.  $A_1 \supseteq_r W \supseteq_r A_2$ . Once the window “slides” and the first element of  $A_2$  expires, we delete the bucket  $A_1$  and renumber the indices so that  $A_2$  becomes the new  $A_1$ ,  $A_3$  becomes the new  $A_2$ , etc. We use the estimated value of bucket  $A_1$  to estimate the value of the current window. The relation between the value of  $f$  on the window and on the first bucket is given by  $(1 - \alpha)f(A_1) \leq f(A_2) \leq f(W) \leq f(A_1)$ .

### 3 A Semi-smooth Estimation of Frequent Elements

In this section we develop an efficient semi-smooth algorithm for finding elements that occur frequently within the window. Let  $n_i$  be the frequency of element  $i \in \{1, \dots, u\}$ , i.e., the number of times  $i$  appears in the window. The *first frequency norm* and the *second frequency norm* of the window are defined by  $L_1 = \sum_{i=1}^u n_i = N$  and  $L_2 = (\sum_{i=1}^u n_i^2)^{\frac{1}{2}}$ . In many previous works, (e.g., [17, 3, 40, 45, 28]) the task of finding heavy-elements is defined using the  $L_1$  norm as follows,

**Definition 3.1 (( $\gamma, \epsilon$ )-approximation of  $L_1$ -heavy hitters).** Output any element  $i \in [u]$  such that  $n_i \geq \gamma L_1$  and no element such that  $n_i \leq (1 - \epsilon)\gamma L_1$ .

Our notion of approximating *frequent elements* is given by Definition 1.1. An equivalent definition which we use in our proof is the following:

**Definition 3.2.** For  $0 < \epsilon, \gamma < 1$ , output all elements  $i \in [u]$  with frequency higher than  $(1 + \epsilon)\gamma L_2$ , and do not output any element with frequency lower than  $(1 - \epsilon)\gamma L_2$ .

Observe that the  $L_2$  approximation is stronger than the above  $L_1$  definition. If an element is heavy in terms of  $L_1$  norm, it is also heavy in terms of the  $L_2$  norm,  $n_i \geq \gamma L_1 = \gamma \sum_j n_j \implies n_i^2 \geq \gamma^2 (\sum_j n_j)^2 \geq \gamma^2 \sum_j n_j^2 = (\gamma L_2)^2$ , while the opposite direction does not apply in general.

In order to identify the frequent elements in the current window, use a variant of the COUNTSKETCH algorithm of Charikar et al. [14], which provides an  $\epsilon$ -approximation (in the unbounded stream model) for the following top-frequent approximation problem.

**Definition 3.3 (( $k, \epsilon$ )-top frequent approximation).** *Output a list of  $k$  elements such that every element  $i$  in the output has a frequency  $n_i > (1 - \epsilon)\phi_k$ , where  $\phi_k$  is the frequency of the  $k$ -th most frequent element in the stream.*

The COUNTSKETCH algorithm guarantees that any element that satisfies  $n_i > (1 + \epsilon)\phi_k$ , appears in the output. This algorithm runs on a stream of size  $n$  and succeeds with probability at least  $1 - \delta$ , and memory complexity of  $O\left(\left(k + \frac{1}{(\epsilon\gamma)^2}\right) \log \frac{n}{\delta}\right)$ , for every  $\delta > 0$ , given that  $\phi_k \geq \gamma L_2$ .

Definition 3.3 and Definition 1.1 do not describe the same problem, yet they are strongly connected. In fact, our method allows solving the frequent elements problem under both definitions, however in this paper we focus on solving the  $L_2$ -frequent-elements problem, as defined by Definition 3.2. In order to do so, we use a variant of the COUNTSKETCH algorithm with specific parameters tailored for our problem (See Appendix A). This variant outputs a list of elements, and is guaranteed to output every element with frequency at least  $(1 + \epsilon')\gamma L_2$  and no element of frequency less than  $(1 - \epsilon')\gamma L_2$ , for an input parameter  $\epsilon'$ .

We stress that COUNTSKETCH is not sufficient on its own to prove Theorem 1.2. The main reason is that this algorithm works in the unbounded stream model, rather than in the sliding window model. Another reason is that it must be tweaked in order not to output false positives. Our solution below makes a use of smooth-histograms to overcome these issues.

### 3.1 Semi-smooth Algorithm for Frequent Elements Approximation

We construct a smooth-histogram for the  $L_2$  norm, and partition the stream into buckets accordingly. It is known that the  $L_2$  property is a  $(\epsilon, \frac{\epsilon^2}{2})$ -smooth function [9]. Using the method of Charikar et al. [14], separately on each bucket, with a careful choice of parameters, we are able to approximate the  $(\gamma, \epsilon)$ -frequent elements problem on a sliding window (Fig. 1).

#### ApproxFreqElements( $\gamma, \epsilon, \delta$ )

1. Maintain an  $(\frac{\epsilon}{2}, \frac{\delta}{2})$ -estimation of the  $L_2$  norm of the window, using a smooth-histogram.
2. For each bucket of the smooth-histogram,  $A_1, A_2, \dots$  maintain an approximated list of the  $k = \frac{1}{\gamma^2} + 1$  most frequent elements, by running  $(\gamma, \frac{\epsilon}{4}, \frac{\delta}{2})$ -COUNTSKETCH<sub>b</sub>. (see COUNTSKETCH<sub>b</sub>'s description in Appendix A).
3. Let  $\hat{L}_2$  be the approximated value of the  $L_2$  norm of the current window  $W$ , as given by the smooth-histogram. Let  $q_1, \dots, q_k \in \{1, \dots, u\}$  be the list of the  $k$  most heavy elements in  $A_1$ , along with  $\hat{n}_1, \dots, \hat{n}_k$  their estimated frequencies, as outputted by COUNTSKETCH<sub>b</sub>.
4. Output any element  $q_i$  that satisfies  $\hat{n}_i > \frac{1}{1+\epsilon}\gamma\hat{L}_2$ .

**Fig. 1.** A semi-smooth algorithm for the frequent elements problem

**Theorem 3.4.** *The semi-smooth algorithm **ApproxFreqElements** (Fig. 1) is a  $(\gamma, O(\epsilon))$ -approximation of the  $L_2$ -frequent elements problem, with success probability at least  $1 - \delta$ . The scheme takes  $O\left(\frac{1}{\gamma^2\epsilon^4} \log N \log \frac{N}{\delta} + \frac{1}{\epsilon^4} \log N \log \frac{1}{\epsilon}\right)$  memory.*

### 3.2 Extensions to Any $L_p$ with $p < 2$

It is easy to see that the same method can be used in order to approximate  $L_p$ -heavy elements for any  $0 < p < 2$ , up to a  $1 \pm \epsilon$  precision. The algorithms and analysis remain the same, except for using a smooth-histogram for the  $L_p$  norm, and changing the parameters by constants.

**Theorem 3.5.** *For any  $p \in (0, 2]$ , there exists a sliding window algorithm that outputs all the elements with frequency at least  $(1 + \epsilon)\gamma L_p$ , and no element with frequency less than  $(1 - \epsilon)\gamma L_p$ . The algorithm succeeds with probability at least  $1 - \delta$  and takes  $\text{poly}(\epsilon^{-1}, \gamma^{-1}, \log N, \log \delta^{-1})$  memory.*

## 4 Semi-smooth Schemes for $\alpha$ -Rarity and Similarity

In this section we extend the method shown above and apply it to other non-smooth functions. In contrast to the smooth  $L_2$  used above, in this section we use a different smooth function to partition the stream, namely the distinct elements count problem. This allows us to obtain efficient semi-smooth approximations for the (non-smooth) *similarity* and  $\alpha$ -rarity tasks.

We begin by stating that counting the number of distinct elements in a stream is smooth. This allows us to partition the stream into a smooth-histogram structure, where each two adjacent buckets have approximately the same number of distinct elements.

**Proposition 4.1.** *Define  $\text{DEC}(A)$  as the number of distinct elements in the stream  $A$ , i.e.,  $\text{DEC}(A) = |A|$ . The function  $\text{DEC}$  is an  $(\epsilon, \epsilon)$ -smooth-function, for every  $0 \leq \epsilon \leq 1$ .*

Another tool we use is *min-wise hash functions* [13,11], used in various algorithms in order to estimate different characteristics of data streams, especially the *similarity* of two streams [13]. Informally speaking, these functions have a meaning of uniformly sampling an element from the stream, which makes them a very useful tool.

**Definition 4.2 (min-hash).** *Let  $\Pi = \{\pi_i\}$  be a family of permutations over  $[u] = \{1, \dots, u\}$ . For a subset  $A \subseteq [u]$  define  $h_i$  to be the minimal permuted value of  $\pi_i$  over  $A$ ,  $h_i = \min_{a \in A} \pi_i(a)$ . The family  $\{h_i\}$  is called  $\epsilon$ -approximated min-wise independent hash functions (or  $\epsilon$ -min-hash) if for any subset  $A \subseteq [u]$  and  $a \in A$ ,  $\Pr_i[h_i(A) = \pi_i(a)] \in \frac{1}{|A|}(1 \pm \epsilon)$ .*

Min-hash functions can be used in order to estimate the similarity of two sets, by using the following lemma,

**Lemma 4.3.** ([11]. See also [21].) *For any two sets  $A$  and  $W$  and an  $\epsilon'$ -min-hash function  $h_i$ , it holds that  $\Pr_i[h_i(A) = h_i(W)] = \frac{|A \cap W|}{|A \cup W|} \pm \epsilon'$ .*

#### 4.1 A Semi-smooth Estimation of $\alpha$ -Rarity

In the following section we present an algorithm that estimates the  $\alpha$ -rarity of a stream (in the sliding window model), i.e., the ratio of elements that appear exactly  $\alpha$  times in the window. The rarity property is known not to be smooth, yet by using a smooth-histogram for distinct elements count, we are able to partition the stream into  $O(\frac{1}{\epsilon} \log N)$  buckets, and estimate the  $\alpha$ -rarity in each bucket.

**Definition 4.4.** *An element  $x$  is  $\alpha$ -rare if it appears exactly  $\alpha$  times in the stream. The  $\alpha$ -rarity measure,  $\rho_\alpha$ , denotes the ratio of  $\alpha$ -rare elements in the entire stream  $S$ , i.e.,  $\rho_\alpha = \frac{|\{x \mid x \text{ is } \alpha\text{-rare in } S\}|}{\text{DEC}(S)}$ .*

Our algorithm follows the method used by [21] to estimate  $\alpha$ -rarity in the unbounded model. The estimation is based on the fact that the  $\alpha$ -rarity is equal to the portion of min-hash functions that their min-value appears exactly  $\alpha$  times in the stream.

However, in order to estimate rarity over sliding windows, one needs to estimate the ratio of min-hash functions of which the min-value appears exactly  $\alpha$  times *within the window*. Our algorithm builds a smooth-histogram for DEC in order to partition the stream into buckets, such that each two consecutive buckets have approximately the same number of distinct elements. In addition, we sample the bucket using a min-wise hash, and count the  $\alpha + 1$  last occurrences of the sampled element  $x_i$  in the bucket. We estimate the  $\alpha$ -rarity of the window by calculating the fraction of min-hash functions of which the appropriate min-value  $x_i$  appears exactly  $\alpha$  times *within the window*. Due to feasibility reasons we use approximated min-wise hashes, and prove that this estimation is an  $\epsilon$ -approximation of the  $\alpha$ -rarity of the current window (up to a pre-specified additive precision). The *semi-smooth* algorithm **ApproxRarity** for  $\alpha$ -rarity is defined in Fig. 2.

#### ApproxRarity( $\epsilon, \delta$ )

1. Randomly choose  $k \frac{\epsilon}{2}$ -min-hash functions  $h_1, h_2, \dots, h_k$ .
2. Maintain an  $(\epsilon, \frac{\delta}{2})$ -estimation of the number of distinct elements by building a smooth histogram.
3. For every bucket instance  $A_j$  of the smooth-histogram and for each one of the hash functions  $h_i, i \in [k]$ 
  - (a) maintain the value of the min-hash function  $h_i$  over the bucket,  $h_i(A_j)$
  - (b) maintain a list  $L_i(A_j)$  of the most recent  $\alpha + 1$  occurrences of  $h_i(A_j)$  in  $A_j$
  - (c) whenever the value  $h_i(A_j)$  changes, re-initialize the list  $L_i(A_j)$ , and continue maintaining the occurrences of the new value  $h_i(A_j)$ .
4. Output  $\hat{\rho}_\alpha$ , the ratio of the min-hash functions  $h_i$ , which has exactly  $\alpha$  active elements in  $L_i(A_1)$ , i.e. the ratio

$$\hat{\rho}_\alpha = |\{i \text{ s.t. } L_i(A_1) \text{ consists exactly } \alpha \text{ active elements}\}|/k .$$

**Fig. 2.** Semi-smooth algorithm for  $\alpha$ -rarity

**Theorem 4.5.** *The semi-smooth algorithm (Fig. 2) is an  $(\epsilon, \delta)$ -approximation for the  $\alpha$ -rarity problem, up to an additive precision. The space complexity is  $\tilde{O}\left(\frac{k}{\epsilon} \alpha \log^2 N\right)$ .*

## 4.2 A Semi-smooth Estimation of Streams Similarity

In this section we present an algorithm for calculating the *similarity* of two streams  $X$  and  $Y$ . As in the case of the rarity, the similarity property is known not to be smooth, however we are able to design a semi-smooth algorithm that estimates it. We maintain a smooth-histogram of the distinct elements count in order to partition each of the streams, and sample each bucket of this partition using a min-hash function. We compare the ratio of sample agreements in order to estimate the similarity of the two streams.

**Definition 4.6.** *The (Jaccard) similarity of two streams,  $X$  and  $Y$  is given by  $S(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$ .*

Recall that for two streams  $X$  and  $Y$ , a reasonable estimation of  $S(X, Y)$  is given by the number of min-hash values they agree on [21]. In other words, let  $h_1, h_2, \dots, h_k$  be a family of  $\epsilon$ -min hash functions and let

$$\hat{S}(X, Y) = |\{i \in [k] \text{ s.t. } h_i(X) = h_i(Y)\}| / k,$$

then  $\hat{S}(X, Y) \in (1 \pm \epsilon)S(X, Y) + \epsilon(1 + p)$ , with success probability at least  $1 - \delta$ , where  $p$  and  $\delta$  are determined by  $k$ . The semi-smooth algorithm **ApproxSimilarity** is rather straightforward and is given in Fig. 3.

### ApproxSimilarity( $\epsilon, \delta$ )

1. Randomly choose  $k$   $\epsilon'$ -min-hash functions,  $h_1, \dots, h_k$ . The constant  $\epsilon'$  will be specified later, as a function of the desired precision  $\epsilon$ .
2. For each stream ( $X$  and  $Y$ ) maintain an  $(\epsilon', \frac{\delta}{2})$ -estimation of the number of distinct elements by building a smooth histogram.
3. For each stream and for each bucket instance  $A_1, A_2, \dots$ , separately calculate the values of each of the min-hash functions  $h_i$ ,  $i = 1 \dots k$ .
4. Let  $A_X$  ( $A_Y$ ) be the first smooth-histogram bucket that includes the current window  $W_X$  ( $W_Y$ ) of the stream  $X$  ( $Y$ ). Output the ratio of hash-functions  $h_i$  which agree on the minimal value, i.e.,

$$\hat{\sigma}(W_X, W_Y) = |\{i \in [k] \text{ s.t. } h_i(A_X) = h_i(A_Y)\}| / k.$$

**Fig. 3.** A semi-smooth algorithm for estimating similarity

**Theorem 4.7.** *The semi-smooth algorithm for estimating similarity (Fig. 3), is an  $(\epsilon, \delta)$ -approximation for the similarity problem, up to an additive precision. The space complexity is  $\tilde{O}\left(k \frac{1}{\epsilon} \log^2 N\right)$ .*

## References

1. Aggarwal, C.C.: Data streams: models and algorithms. Springer, New York (2007)
2. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences* 58(1), 137–147 (1999)
3. Arasu, A., Manku, G.S.: Approximate counts and quantiles over sliding windows. In: PODS 2004, pp. 286–296 (2004)
4. Bandi, N., Agrawal, D., Abbadi, A.E.: Fast algorithms for heavy distinct hitters using associative memories. In: ICDSC 2007, p. 6 (2007)
5. Bar-Yossef, Z., Jayram, T.S., Kumar, R., Sivakumar, D., Trevisan, L.: Counting distinct elements in a data stream. In: Rolim, J.D.P., Vadhan, S.P. (eds.) RANDOM 2002. LNCS, vol. 2483, pp. 1–10. Springer, Heidelberg (2002)
6. Bar-Yossef, Z., Jayram, T.S., Kumar, R., Sivakumar, D.: An information statistics approach to data stream and communication complexity. In: FOCS 2002, pp. 209–218 (2002)
7. Bar-Yossef, Z., Kumar, R., Sivakumar, D.: Reductions in streaming algorithms, with an application to counting triangles in graphs. In: SODA 2002, pp. 623–632 (2002)
8. Bhuvanagiri, L., Ganguly, S., Kesh, D., Saha, C.: Simpler algorithm for estimating frequency moments of data streams. In: SODA 2006, pp. 708–713 (2006)
9. Braverman, V., Ostrovsky, R.: Smooth histograms for sliding windows. In: FOCS 2007, pp. 283–293 (2007)
10. Braverman, V., Gelles, R., Ostrovsky, R.: How to catch  $L_2$ -heavy-hitters on sliding windows (2010), <http://arxiv.org/abs/1012.3130>
11. Broder, A.Z., Charikar, M., Frieze, A.M., Mitzenmacher, M.: Min-wise independent permutations. *Journal of Computer and System Sciences* 60(3), 630–659 (2000)
12. Broder, A.Z., Glassman, S.C., Manasse, M.S., Zweig, G.: Syntactic clustering of the web. *Computer Networks and ISDN Systems* 29(8-13), 1157–1166 (1997)
13. Broder, A.: On the resemblance and containment of documents. In: Proceedings of the Compression and Complexity of Sequences 1997, pp. 21–29 (1997)
14. Charikar, M., Chen, K., Farach-Colton, M.: Finding frequent items in data streams. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 693–703. Springer, Heidelberg (2002)
15. Cohen, E.: Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences* 55(3), 441–453 (1997)
16. Cohen, E., Strauss, M.J.: Maintaining time-decaying stream aggregates. *Journal of Algorithms* 59(1), 19–36 (2006)
17. Cormode, G., Muthukrishnan, S.: An improved data stream summary: The count-min sketch and its applications. In: Farach-Colton, M. (ed.) LATIN 2004. LNCS, vol. 2976, pp. 29–38. Springer, Heidelberg (2004)
18. Cormode, G., Hadjieleftheriou, M.: Finding frequent items in data streams. Proc. VLDB Endow. 1(2), 1530–1541 (2008)
19. Cormode, G., Korn, F., Muthukrishnan, S., Srivastava, D.: Finding hierarchical heavy hitters in data streams. In: VLDB 2003, pp. 464–475 (2003)
20. Cormode, G., Muthukrishnan, S.: What's hot and what's not: tracking most frequent items dynamically. ACM Trans. Database Syst. 30(1), 249–278 (2005)
21. Datar, M., Muthukrishnan, S.: Estimating rarity and similarity over data stream windows. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 323–334. Springer, Heidelberg (2002)

22. Datar, M., Gionis, A., Indyk, P., Motwani, R.: Maintaining stream statistics over sliding windows (extended abstract). In: SODA 2002, pp. 635–644 (2002)
23. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Frequency estimation of internet packet streams with limited space. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 348–360. Springer, Heidelberg (2002)
24. Estan, C., Varghese, G.: New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. ACM Trans. Comput. Syst. 21(3), 270–313 (2003)
25. Flajolet, P., Martin, G.N.: Probabilistic counting. In: FOCS 1983, pp. 76–82 (1983)
26. Gibbons, P.B., Tirthapura, S.: Estimating simple functions on the union of data streams. In: SPAA 2001, pp. 281–291 (2001)
27. Golab, L., DeHaan, D., Demaine, E.D., López-Ortiz, A., Munro, J.I.: Identifying frequent items in sliding windows over on-line packet streams. In: IMC 2003, pp. 173–178 (2003)
28. Hung, R.Y.S., Ting, H.F.: Finding heavy hitters over the sliding window of a weighted data stream. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) LATIN 2008. LNCS, vol. 4957, pp. 699–710. Springer, Heidelberg (2008)
29. Hung, R.Y., Lee, L.K., Ting, H.: Finding frequent items over sliding windows with constant update time. Information Processing Letters 110(7), 257–260 (2010)
30. Indyk, P.: A small approximately min-wise independent family of hash functions. In: SODA 1999, pp. 454–456 (1999)
31. Indyk, P.: Heavy hitters and sparse approximations, lecture notes (2009), <http://people.csail.mit.edu/indyk/Rice/lec4.pdf>
32. Indyk, P., Woodruff, D.: Optimal approximations of the frequency moments of data streams. In: STOC 2005, pp. 202–208 (2005)
33. Jin, C., Qian, W., Sha, C., Yu, J.X., Zhou, A.: Dynamically maintaining frequent items over a data stream. In: CIKM 2003, pp. 287–294 (2003)
34. Kane, D.M., Nelson, J., Woodruff, D.P.: An optimal algorithm for the distinct elements problem. In: PODS 2010, pp. 41–52 (2010)
35. Karp, R.M., Shenker, S., Papadimitriou, C.H.: A simple algorithm for finding frequent elements in streams and bags. ACM Trans. Database Syst. 28, 51–55 (2003)
36. Lee, L.K., Ting, H.F.: A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In: PODS 2006, pp. 290–297 (2006)
37. Manku, G.S., Motwani, R.: Approximate frequency counts over data streams. In: VLDB 2002, pp. 346–357 (2002)
38. Open problems in data streams and related topics. IITK Workshop on Algorithms for Data Streams 2006 (2006), compiled and edited by McGregor, A.
39. Metwally, A., Agrawal, D., El Abbadi, A.: Efficient computation of frequent and top-k elements in data streams. In: Eiter, T., Libkin, L. (eds.) ICDT 2005. LNCS, vol. 3363, pp. 398–412. Springer, Heidelberg (2005)
40. Muthukrishnan, S.: Data streams: Algorithms and applications. Now Publishers Inc. (2005)
41. Nie, G., Lu, Z.: Approximate frequency counts in sliding window over data stream. In: Canadian Conference on Electrical and Computer Engineering, pp. 2232–2236 (2005)
42. Saks, M., Sun, X.: Space lower bounds for distance approximation in the data stream model. In: STOC 2002, pp. 360–369 (2002)
43. Sen, S., Wang, J.: Analyzing peer-to-peer traffic across large networks. In: IMW 2002, pp. 137–150. ACM (2002)

44. Tirthapura, S., Woodruff, D.P.: A general method for estimating correlated aggregates over a data stream. In: International Conference on Data Engineering, pp. 162–173 (2012)
45. Zhang, L., Guan, Y.: Frequency estimation over sliding windows. In: International Conference on Data Engineering, pp. 1385–1387 (2008)

## A The COUNTSKETCH<sub>b</sub> Algorithm

In this section we describe the COUNTSKETCH<sub>b</sub> algorithm and several of its properties. Let us sketch the details of the COUNTSKETCH algorithm as defined in [14]. COUNTSKETCH is defined by three parameters  $(t, b, k)$  such that the algorithm takes space  $O(tb + k)$ , and if  $t = O(\log \frac{n}{\delta})$  and  $b \geq \max(8k, 256 \frac{L_2}{\epsilon^2 \phi_k^2})$  then the algorithm outputs any element with frequency at least  $(1 + \epsilon)\phi_k$ , except with probability  $\delta$ .  $\phi_k$  is the frequency of the  $k$ th-heavy element, and  $L_2$  is the  $L_2$ -frequency norm of the entire ( $n$ -element) stream. The algorithm works by computing, for each element  $i$ , an approximation  $\hat{n}_i$  of its frequency. The scheme guarantees that with high probability, for every element  $i$ ,  $|\hat{n}_i - n_i| < 8 \frac{L_2(S)}{\sqrt{b}}$  (see Lemma 4 in [14]).

For  $0 < \epsilon', \gamma, \delta \leq 1$  define  $(\gamma, \epsilon', \delta)$ -COUNTSKETCH<sub>b</sub> by setting  $k = \frac{1}{\gamma^2} + 1$  and  $b = \frac{256}{\gamma^2 \epsilon'^2}$  in COUNTSKETCH (the parameter  $t$  remains as in the original scheme). The choice of  $k$  follows from the following known fact.

**Lemma A.1.** *There are at most  $\frac{1}{\gamma^2}$  elements with frequency higher than  $\gamma L_2$ .*

Setting  $k = \frac{1}{\gamma^2} + 1$  ensures that the output list is large enough to contain all the elements with frequency  $\gamma L_2$  or more. However, COUNTSKETCH<sub>b</sub> does not guarantee anymore to output all the elements with frequency higher than  $(1 + \epsilon')\phi_k$  and no element of frequency less than  $(1 - \epsilon')\phi_k$  (Lemma 5 of [14]), since the value of  $b$  might not satisfy the conditions of that lemma.

We can still follow the analysis of [14] and claim that the frequency approximation of each element is still bounded (Lemma 4 of [14]),

**Lemma A.2.** *With probability at least  $1 - \delta$ , for all elements  $i \in [u]$  in the stream  $S$ ,  $|\hat{n}_i - n_i| < 8 \frac{L_2(S)}{\sqrt{b}} < \frac{1}{2} \gamma \epsilon' L_2(S)$  where  $\hat{n}_i$  is the approximated frequency of  $i$  calculated by COUNTSKETCH<sub>b</sub>, and  $n_i$  is the real frequency of the element  $i$ .*

**Proposition A.3.** *The  $(\gamma, \epsilon', \delta)$ -COUNTSKETCH<sub>b</sub> algorithm outputs all the elements whose frequency is at least  $(1 + \epsilon')\gamma L_2(S)$ .*

# Time/Memory/Data Tradeoffs for Variants of the RSA Problem

Pierre-Alain Fouque<sup>1</sup>, Damien Vergnaud<sup>2</sup>, and Jean-Christophe Zapalowicz<sup>3</sup>

<sup>1</sup> University of Rennes 1

[pierre-alain.fouque@ens.fr](mailto:pierre-alain.fouque@ens.fr)

<sup>2</sup> École normale supérieure, Paris, France\*

<sup>3</sup> INRIA Rennes

[jean-christophe.zapalowicz@inria.fr](mailto:jean-christophe.zapalowicz@inria.fr)

**Abstract.** In this paper, we study the security of the Micali-Schnorr pseudorandom number generator. The security of this cryptographic scheme is based on two computational problems which are variants of the RSA problem. The RSA problem essentially aims at recovering the plaintext from a random ciphertext. In the analysis of the Micali-Schnorr pseudorandom generator, we are interested in instances of this problem where the plaintext is small and where the ciphertext is not entirely known. We will describe time / memory tradeoff techniques to solve these hard problems which provides the first analysis of this pseudorandom generator 25 years after its publication.

**Keywords:** Micali-Schnorr generator, Time/Memory/Data tradeoff.

## 1 Introduction

In this paper we study two cryptographic computational problems related to the RSA problem. Given a modulus  $N$ , product of two large prime numbers, and an odd exponent  $e$ , coprime to  $\varphi(N)$  the order of the multiplicative group  $\mathbb{Z}_N^*$ , the RSA problem consists in recovering the plaintext  $m \in \mathbb{Z}_N^*$  from a random ciphertext  $c = m^e \bmod N$ . The variants we look at consider particular instances of this problem where the plaintext is small or where the plaintext is small and only a part of the ciphertext is known. These two problems appear to be related to the security of a pseudorandom generator proposed by Micali and Schnorr [21].

A pseudorandom generator is a deterministic polynomial time algorithm that expands short seeds (made of truly random bits) into longer bit sequences, whose distribution cannot be distinguished from uniformly random bits by a computationally bounded algorithm. Pseudorandom number generators are probably the most basic cryptographic primitive: they are widely used for block ciphers, public-key encryption, digital signatures, keystream generation and as passwords sources. It is well-known that pseudorandom generators exist if and only if one-way functions exist [17] (though this generic construction is highly inefficient).

---

\* ENS, CNRS & INRIA – UMR 8548.

The first practical generator with proven security was proposed by Blum and Micali [6]. The Blum-Micali generator outputs only one bit per iteration – which costs one exponentiation modulo a large prime  $p$  – and its security is based on the intractability of the discrete logarithm problem in  $\mathbb{Z}_p^*$ . It has been shown (e.g. [18]) that the generator remains secure if one outputs  $O(\log \log p)$  bits per iteration. Another line of pseudorandom generators is based on factoring-like assumptions (e.g. [5,21,24]). The BBS generator was introduced by Blum, Blum and Shub in [5] and proven secure under the assumption that deciding quadratic residuosity modulo a composite Blum<sup>1</sup> integer  $N$  is hard. The generator works by repeatedly squaring modulo  $N$  a random seed in  $\mathbb{Z}_N^*$  and outputs (at each iteration) the least significant bit of the current value. Similarly, the RSA generator works by iterating the RSA encryption mapping  $v \mapsto v^e \bmod N$  (for a public RSA modulus  $N$  and a public exponent  $e$  coprime to  $\varphi(N)$ ) on a secret random initial seed value  $v_0 \in \mathbb{Z}_N^*$  to compute the intermediate state values  $v_{i+1} = v_i^e \bmod N$  (for  $i \in \mathbb{N}$ ) and outputting the least significant bit of the state value at each iteration. In [1] Alexi *et al.* showed that one can output up to  $O(\log \log N)$  bits per iteration of the BBS generator and the RSA generator. The actual number of bits that can be output depends on the concrete parameters adopted but the generators are considered too slow for most applications.

Another line of number-theoretic pseudorandom generators sacrifices provable security for efficiency. Gennaro [15] suggested a discrete-logarithm based generator that outputs  $O(\log p)$  bits per modular exponentiation in  $\mathbb{Z}_p^*$  but its security is based on a strong and not so well-studied “discrete logarithm with short exponents” assumption. Steinfeld, Pieprzyk and Wang [24] showed, that assuming the hardness of a strong variant of the RSA inversion problem modulo the integer  $N$ , one can securely output as much as  $(1/2 - 1/e) \log N$  bits in the RSA generator. On the other hand, Herrmann and May [20] showed heuristically (using Coppersmith methods [10,11]) that an output of  $(1 - 1/e) \log N$  most significant bits per iteration allows for efficient recovery of the whole sequence.

Micali and Schnorr [21] proposed a variant of the RSA generator that on a secret random initial seed value  $x_0 \in \mathbb{Z}_N^*$  computes the intermediate values  $v_i = x_i^e \bmod N$  and outputs, for some  $k \in \mathbb{N}$ , the  $k$  least significant bits of  $v_i$ . But the successor  $x_{i+1}$  of  $x_i$  is formed from a separate part of  $v_i$ , the remaining most significant bits (contrary to the *incestuous* RSA generator where  $x_{i+1} = v_i$ ). The security of Micali-Schnorr pseudorandom generator relies on the (strong) assumption that the distribution of  $x^e \bmod N$  for random  $k$ -bit integers is indistinguishable from the uniform distribution on  $\mathbb{Z}_N^*$ . The generator is insecure if  $(1 - 1/e) \log N$  least significant bits are output per iteration but no better attack was proposed since its proposal 25 years ago. It remains open to know what is the maximum quantity of information that can be output per iteration allowing the generator to be efficient but still secure against potential attackers.

**Our Techniques.** As dynamic programming, time / memory tradeoffs is a well-known technique to reduce the time complexity of a problem using memory. Shamir and Schroeppel in [23] have described such algorithms for specific

---

<sup>1</sup>  $N$  is a Blum integer if  $N = pq$  with  $p$  and  $q$  primes and  $p, q \equiv 3 \pmod{4}$ .

NP-complete problems such as knapsack problems. In cryptography, this technique has been used many times to analyze the security of symmetric primitives such as block ciphers or stream ciphers and some computational problems such as the baby-step giant-step algorithm to compute discrete logarithms. Basically, some computations can be done independently of other resources. For instance, using the public key the adversary can precompute some values and store a small fraction of these values in the offline phase. Then, the adversary gets some ciphertexts and his goal can be to recover the secret key.

In [19] Hellman described a technique to invert random looking functions. This technique has been rigorously studied in [13] by Fiat and Naor to work for any functions and rigorous lower bounds have been given in [3] by Barkan, Biham and Shamir. Oeschlin in [22] described a variant of Hellman tradeoff, but this variant has been shown equivalent to Hellman tradeoff by Barkan *et al.* since many heuristics can be applied to Hellman technique. Finally, Babbage [2] and Golic [16], then Biryukov and Shamir [4] presented tradeoff for stream cipher by using more or less data. This resource is a crucial parameter in cryptanalysis and it is important to present attacks using as low data complexity as possible.

**Our Contributions.** In this paper, we use time/memory/data tradeoff techniques to propose algorithms for two computational problems related to the security of the Micali-Schnorr pseudorandom generator. The algorithms are decomposed into two phases: the preprocessing one where the attacker constructs large hash tables using the structure of the focused cryptosystem, and the real-time phase where it uses the data produced by the cryptosystem and the hash tables to retrieve the secrets. The three tradeoffs algorithms we describe are similar to the tradeoffs for stream ciphers. However, in order to construct such algorithms, we need to specify the function  $f$  we used. For stream ciphers, the main idea is to execute from a hidden state the generator in order to have at least  $\log S$  bits of output if the state is of size  $S$ . Here, we decide to truncate the output value. It is a bit weird to define  $f$  in such a way since the iteration of such functions is no more related to the iteration of the generator. However, the only things we need is to cover the space in such a way that the inversion will be possible. This choice of function  $f$  is suitable for Micali-Schnorr generator but does not work for the BBS or the RSA generator. Moreover, in order to prove that the many Hellman tables algorithm works (our third algorithm), we need to prove that each table uses an independent function. We provide such claim in the analysis of the third algorithm. Indeed, this independence assumption is in fact the tricky part of the analysis and Hellman paper relies on heuristic in order to provide lower bounds on the time complexity of his scheme. Using a computational argument we prove that the considered functions are independent.

Our algorithms do not contradict the strong assumption used for the Micali-Schnorr pseudorandom generator. They can be applied even though only a small part of the generator is output at each iteration. Moreover, we will show that once one value is recovered using the algorithms we describe for the first problem, then we are also able to retrieve the seed by using another time/memory tradeoff. Finally, even if our algorithms beating the bound remain exponential, we achieve

to decrease the constant and that can be very interesting in cryptography (for example, in the case of the factorization).

**Organization of the Paper.** In Section 2, we present the first problem we look at and basics about the Micali-Schnorr pseudorandom generator. We explain why the problem is easy for some small parameters. In Section 3, we describe three time/memory algorithms for solving the first problem using different tradeoffs. In Section 4, we show other tradeoffs to recover the seed of the generator.

## 2 Micali-Schnorr Pseudorandom Generator

The Micali-Schnorr pseudorandom generator is defined by the recursive sequence ( $v_i = x_{i-1}^e \bmod N$ ) for  $i \geq 1$ , with  $(e, N)$  the RSA public key,  $x_0 \in [0, 2^r[$  the secret seed of size<sup>2</sup>  $r$  and  $v_i = 2^k x_i + w_i$ . At each iteration, this generator outputs the  $k$  least significant bits of  $v_i$ , denoted by  $w_i$ . In addition, denoting  $n$  the size of the modulus  $N$ , only  $x_i$  of size  $r = n - k$ , unknown, is reused for the next iteration. Since the generator outputs  $O(k/\log e)$  bits per multiplication, one wants  $k$  to be as large as possible and  $e$  to be as small as possible.

This pseudorandom generator is proven secure under the following assumption:

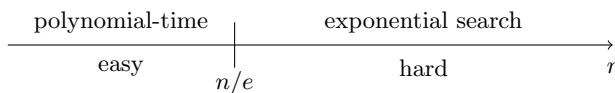
**Assumption 1.** *The distribution of  $x^e \bmod N$  for random  $r$ -bit integers is indistinguishable by all polynomial-time statistical tests from the uniform distribution of elements of  $\mathbb{Z}_N^*$ .*

**Description of the Problem.** Let  $(e, N)$  the RSA public key with  $N$  of size  $n$ . Using the equality  $v_i = 2^k x_i + w_i$  where  $v_i \in \mathbb{Z}_N$ ,  $w_i \in [0, 2^k)$  and  $x_i \in [0, 2^r)$ , we consider the recurrence sequence

$$\forall i \geq 1, \quad v_i = x_{i-1}^e \bmod N \quad (1)$$

Given  $(e, N, r)$ ,  $\{w_1, \dots, w_j\}$  with  $j \in \mathbb{N}$ , the problem consists in retrieving one value  $x_c$  with  $c \in \{0, \dots, j-1\}$ .

For an attacker, finding one of the values  $x_i$  using some iterations of the Micali-Schnorr pseudorandom generator will lead to infer its next outputs. The difficulty of the above problem depends highly on the value of  $r$ . Figure 1 sums up this hardness, with a transition value equal to  $n/e$ . We first explain why it is easy to solve the problem when the size of  $r$  is less than  $n/e$ .



**Fig. 1.** Difficulty of the problem depending of the value  $r$

---

<sup>2</sup> Throughout the paper, the *size* of an integer is (an upper-bound of) its bit-size.

**Theorem 1.** Suppose that the value  $x_0$  of size  $r$  is odd. If  $r \leq n/e$ , given  $(e, N)$  and  $w_1$ , there exists a polynomial-time algorithm which retrieves the value  $x_0$ .

*Proof.* If  $r \leq n/e$ , the modular reduction is not performed in Equation 1, so  $v_1 = x_0^e$  over the integers and using  $v_1 = 2^k x_1 + w_1$ , one has the following modular equation:

$$x_0^e = w_1 \pmod{2^k}$$

where all the values except  $x_0$  are known. We now use the well-known Hensel's lifting lemma to retrieve this secret value.

**Lemma 1 (Hensel's lifting lemma).** Let  $p$  be a prime and  $c$  be a positive integer. One denotes  $f$  a polynomial having a root  $x$  modulo  $p^c$  which satisfies:

$$f(x) = 0 \pmod{p^c} \quad \text{and} \quad f'(x) \neq 0 \pmod{p}$$

Then, one can lift  $x$  to obtain an unique nontrivial root  $x^* \in [0, p^{c+1})$  verifying:

$$f(x^*) = 0 \pmod{p^{c+1}} \quad \text{and} \quad x^* = x \pmod{p^c}$$

With Lemma 1, by using  $f(x) = x^e - w_1$ , one can reconstruct bit per bit  $x_0$  looking at the powers of 2. The value  $x^*$  can be efficiently computed by  $x^* = x + \lambda \cdot 2^c$  where  $\lambda = -\frac{f(x)}{2^c} \cdot (f'(x))^{-1} \pmod{2}$ .  $\square$

Note that if the value  $x_0$  is even, one loses the uniqueness of the lift. However, computing  $x^e - w_1 \pmod{2^k}$  for each candidate  $x$  of size  $r$  can suffice to retrieve this value; else one tests another output  $w_i$  of the generator.

Another possibility to retrieve the seed consists in raising  $w_1$  to the power  $e^{-1} \pmod{2^{k-1}}$  (notice that  $e$  is odd). However the complexity of Hensel lifting is linear in the size of the root, contrary to this exponentiation.

To avoid this simple algorithm but to remain efficient, i.e to output a maximum of bits per iteration, the parameter  $k$  has to be smaller than  $\lfloor n(1 - \frac{1}{e}) \rfloor$ . Finally, it seems hard to find a polynomial-time algorithm if  $r > n/e$ , for example by using Coppersmith techniques, which are techniques bases on lattice reduction to find small modular roots.

### 3 Solving the Problem Using Time/Memory/Data Tradeoffs

For now, we consider the problem in the case where  $r$  is larger than  $n/e$  and we will present three similar algorithms that use different tradeoffs in order to solve the problem. These algorithms use the fact that only the hidden information, i.e the value  $x_i$  of a relatively small size  $r$ , is recycled for the next iteration contrary to some other pseudorandom generators as the BBS or the RSA ones. We denote the five key parameters as follows:

- $2^r$  represents the cardinality of the search space.
- $P$  represents the time required by the preprocessing phase of the algorithm.
- $M$  represents the quantity of access memory required for the algorithm.
- $T$  represents the time required by the online phase of the algorithm.
- $D$  represents the quantity of data required for the algorithm.

### 3.1 First Algorithm

The first algorithm is quite simple to explain and to implement but not really efficient. The preprocessing phase consists in storing the couples  $(x, \text{LSB}_k(x^e \bmod N))$  for some different values of  $x$  in a hash table. During the online phase, one tests for each value  $w_i$  if it appears in the hash table. For example, it will work by taking  $M = 2^{r/3}$  and  $D = T = 2^{2r/3}$  or even  $M = T = D = 2^{r/2}$ . The proof is given in the full version of this paper.

**Theorem 2.** *Given  $(e, N)$  and  $D$  consecutive values  $w_1, \dots, w_D$ , there exists an algorithm which retrieves one of the values  $x_0, \dots, x_{D-1}$  in time  $T$ , by using  $M$  random access memory such that  $TM = O(2^r)$  with  $D = O(T)$ .*

### 3.2 Second Algorithm Using one Hellman's Table

The two next algorithms are based on [19,4]. Hellman then Biryukov and Shamir have proposed different attacks using tradeoffs for breaking block ciphers and stream ciphers. We define a special function in order to apply these attacks for solving our problem, and thus for the Micali-Schnorr pseudorandom generator. This second algorithm gives the same tradeoff as the first one, but need less data: for  $M = 2^{r/3}$  and  $T = 2^{2r/3}$ , it just requires a bit more than  $2^{r/3}$  data.

**Theorem 3.** *Given  $(e, N)$  and  $D$  consecutive values  $w_1, \dots, w_D$ , there exists an algorithm which retrieves one of the values  $x_0, \dots, x_{D-1}$  in time  $T$ , by using  $M$  random access memory such that  $TM = O(2^r)$  with  $T \leq D^2$ .*

*Proof.* **Algorithm.** Let  $f$  be the function defined by  $f(x) = \text{LSB}_r(x^e \bmod N)$  where  $\text{LSB}_r(x)$  represents the  $r$  least significant bits of  $x$ . The preprocessing phase consists in computing for  $m$  random different values  $x_0^1, \dots, x_0^m$  the values  $f^t(x_0^1), \dots, f^t(x_0^m)$  with  $m, t \in \mathbb{N}$  and where  $f^t$  means that the function  $f$  is iterated  $t$  times. The construction of a hash table containing the  $f^t(x_0^i)$  as keys and the  $x_0^i$  as associated values, for  $i \in \{1, \dots, m\}$ , concludes this phase.

The algorithm in online phase works as follows:

1. One selects a known value  $w_j$  for  $j > 0$ .
2. One considers only the  $r$  least significant bits of  $w_j$ , denoted by  $z_j$ .
3. For  $i \in \{0, \dots, t\}$ , one tests if  $f^i(z_j)$  is a key of the hash function. If the  $t + 1$  tests fail, one selects the next known value and restarts the algorithm.
4. If a test succeeds, denoting the associated value  $x_0^c$ , one has:

$$f^{t-i}(x_0^c) = z_j = f(\underbrace{f^{t-i-1}(x_0^c)}_X)$$

$X$  is a value of size  $r$  that corresponds with high probability to the hidden part of the generator at the previous iteration. A simple verification consists of the computation of the value  $X^e \bmod N$ .

**Table 1.** Computation of our algorithm using a hash table

| Value    | Hellman's Matrix for our algorithm                                                                                                                            | Key          |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|
| $x_0^1$  | $\xrightarrow{f} f(x_0^1) \xrightarrow{f} \dots \xrightarrow{f} f^{t-1}(x_0^1) \xrightarrow{f}$                                                               | $f^t(x_0^1)$ |
| $\vdots$ |                                                                                                                                                               | $\vdots$     |
| $x_0^c$  | $\underbrace{\xrightarrow{f} \dots \xrightarrow{f} X}_{\text{step 4}} \xrightarrow{f} \underbrace{z_j \xrightarrow{f} \dots \xrightarrow{f}}_{\text{step 3}}$ | $f^i(z_j)$   |
| $\vdots$ |                                                                                                                                                               | $\vdots$     |
| $x_0^m$  | $\xrightarrow{f} f(x_0^m) \xrightarrow{f} \dots \xrightarrow{f} f^{t-1}(x_0^m) \xrightarrow{f}$                                                               | $f^t(x_0^m)$ |

Table 1 gives an overview of the algorithm by manipulating the hash table and using the function  $f$ .

**Complexity.** The number of different values in this table can be estimated as follows (the end value of each chain is not counted):

$$E(\#\{f^j(x_0^i), 1 \leq i \leq m, 0 \leq j < m\}) = \sum_{i=1}^m \sum_{j=0}^{t-1} Pr[A_{i,j}]$$

where  $A_{i,j}$  the event  $[f^j(x_0^i) \notin \{f^{j'}(x_0^{i'}), i' < i \text{ or } j' < j\}]$ . Note that  $A_{i,j} \subseteq A_{i,j-1}$  (since  $f^j(x_0^i) = f^{j-1}(x_0^i)$ ). Moreover, we have the following property:

$$Pr[A_{i,j}|A_{i,j-1}] \geq 1 - \frac{it}{2^r} \Rightarrow Pr[A_{i,j}] \geq \left(1 - \frac{it}{2^r}\right)^{j+1}$$

Hence, the probability  $p$  that the value we search is in the Hellman's table is greater than  $2^{-r} \sum_{i=1}^m \sum_{j=0}^{t-1} \left(1 - \frac{it}{2^r}\right)^{j+1}$ .

By denoting  $D$  the number of known values of the recurrence, the time required by the preprocessing phase of the algorithm  $P$  is equal to  $O(mt)$ , the memory  $M$  and the time of the algorithm  $T$  are defined by  $M = O(m)$  and  $T = O(Dt)$ . For each known value of the recurrence, one has a probability  $p$  depending on the size of the table to success, and thus we need  $Dp = O(1)$ . If  $mt^2 \ll 2^r$  then  $p \approx 2^{-r}mt$ . Consequently we obtain the tradeoff  $Dt \cdot m = O(2^r)$ , i.e  $TM = 2^r$  with  $T \leq D^2$  (due to  $mt^2 \ll 2^r$ ).

As the table gets larger, some chains will eventually collide and merge due to the birthday paradox. So it may be preferable to use many small tables, that is the next attack. Finally note that, at step 4, each value  $f^i(z_j)$  may have multiple predecessors, hence there is a small probability that  $f^{t-i}(x_0^c)$  will not be equal to  $z_j$ . In this case, one tries an other output but it is clear that these “false alarms” will increase the complexity by only a small constant factor.  $\square$

### 3.3 Third Algorithm Using Many Hellman Tables

This last algorithm which uses more tables, proposes then another repartition between the memory, the time and the data. For example, for  $M = 2^{5r/8}$ ,  $T = 2^{r/2}$  and  $D = 2^{r/8}$ , this tradeoff is preferable compared to the first two algorithms.

This set of tables covers a larger fraction of the possible output values and consequently, the online phase need less data. Each table requires a specific function and, in order to cover different independent output values, the functions need to be independent. In [19], Hellman rigorously calculated a lower bound on the expected coverage of images by a *single* table which is essentially the same analysis we did in the previous algorithm. However, the analysis for the full scheme (with *many* tables) is highly heuristic and is based on the unjustifiable assumption that many simple variants of  $f$  are independent of each other. Fiat and Naor in [13] propose to use  $k$ -wise independent functions in order to propose an algorithm to invert *any* function, while Hellman assumes that the function is random. In order to replace the heuristic, one could think of using independent functions for each table by computing  $g_i = h_i \circ f$ , where  $\{h_i\}_i$  is a family of  $k$ -wise independent functions. The main drawback is that the number of such functions we need is exponential and it is not easy to construct such functions. Here, we want to avoid Hellman heuristic while similar heuristic could be made. For instance, we could define many functions by considering any  $r$  bits among the  $n - r$  output bits which will give us  $\binom{n-r}{r}$  different functions. However, many functions will have the same subset of bits and we cannot assume independence between them. The analysis of the algorithm is based on the following hypothesis:

**Assumption 2.** Denoting  $f(x) = LSB_r(x^e \bmod N)$ , the distribution of  $f(x)$  for random  $r''$ -bit integers ( $r'' \geq r$ )  $x$  is indistinguishable by all polynomial-time statistical tests from the uniform distribution of integers in  $[0, 2^r]$ .

**Theorem 4.** Given  $(e, N)$  and  $D$  consecutive values  $w_1, \dots, w_D$ , there exists an algorithm which retrieves one of the values  $x_0, \dots, x_{D-1}$  in time  $T$ , by using  $M$  random access memory such that  $TM^2D^2 = O(2^{2r})$  with  $D^2 \leq T \leq 2^r$ .

*Proof.* **Algorithm.** The third algorithm is similar to the second one but, instead of using a single table, one uses  $\ell = t/D$  hash tables of size  $mt$  (assuming that  $t > D$ ). First, one has to find which table covers the output value. Then one applies the second algorithm. Consequently, the search of the table requires to look for each value in all tables in parallel.

**Complexity.** Using the same analysis as for the second algorithm, we know that we cover a fraction  $mt/2^r$  of the output values with one table. Now, using  $\ell$  tables, we want to prove that the number of output values we cover is  $mt\ell$ . To prove such result, we have to solve the independence problem, namely that to describe independent functions for each table so that we are still able to invert  $f$ . First of all, the whole output of the  $n - r$  least significant bits of  $x^e \bmod N$  can be used. But this only allows us to construct a constant number of functions. Our second idea is to use the fact that  $f$  is a random function or that its outputs are indistinguishable from the uniform distribution (Assumption 2). By using

$\ell$  random and independent values  $z_i \in [0, 2^{r'}]$ , we can define  $\ell$  functions as  $g_i(x) = f(x + z_i \cdot 2^r)$  for  $i \in \{1, \dots, \ell\}$ . We claim that this set of functions is independent, otherwise assumption 2 will be wrong for  $r'' = r + r'$ .

Using the same notations as in the previous proof, the probability  $p$  that the value we search is in one of the  $\ell$  Hellman's tables is greater than  $1 - (1 - 2^{-r} \sum_{i=1}^m \sum_{j=0}^{t-1} (1 - \frac{it}{2^r})^{j+1})^\ell$  and  $p \approx 2^{-r} m t \ell$  if  $m t^2 \ll 2^r$ . We clearly have  $M = O(m\ell)$ ,  $T = O(Dt\ell)$  and  $P = O(mtl)$ . For  $\ell = t/D$ , we obtain the tradeoff  $TM^2D^2 = O(2^{2r})$  with  $D^2 \leq T \leq 2^r$ .  $\square$

*Remark 1.* This tradeoff is less constraining than Hellman tradeoff: we only need that one value is in one table and not that a particular value.

## 4 Inverting RSA for Small Plaintext Problem

By using one of the previous algorithms, one knows the value of a hidden part of the generator denoted  $x_i$  for  $i \geq 0$ . We now present two different ways to invert the Micali-Schnorr generator, i.e to retrieve the secret seed  $x_0$ .

**Description of the Problem.** Let  $(e, N)$  be an RSA public key with  $N$  of size  $n$  and an integer  $r \leq n$ . Given  $(N, e, r)$  and  $y = x^e \bmod N$  for  $x \in [0, 2^r]$ , the problem consists in recovering  $x$ .

*Remark 2.* This problem is well-known to be solvable in polynomial time when  $r \leq n/e$  since as before the equality holds over the integers.

### 4.1 Multipoint Evaluation of Univariate Polynomials

Let  $P(x) \in \mathbb{Z}_N[x]$  be a polynomial of degree less than  $n = 2^k$ . The multipoint evaluation problem is the task of evaluating  $P$  at  $n$  distinct points  $\alpha_0, \dots, \alpha_{n-1} \in \mathbb{Z}_N$ . Using Horner's rule, it is easy to propose a solution that uses  $O(n^2)$  addition and multiplication in  $\mathbb{Z}_N$  but it is well-known that one can propose an algorithm with quasi-linear complexity  $\tilde{O}(n)$  operations in  $\mathbb{Z}_N$  using a divide-and-conquer approach [8,14].

Let  $P_0 = \prod_{\ell=0}^{n/2-1} (x - \alpha_\ell)$  and  $P_1 = \prod_{\ell=n/2}^{n-1} (x - \alpha_\ell)$  and let us define  $R_0 = P \bmod P_0$  and  $R_1 = P \bmod P_1$ . We have  $R_0(\alpha_i) = P(\alpha_i)$  for all  $i \in \{0, \dots, n/2-1\}$  and  $R_1(\alpha_i) = P(\alpha_i)$  for all  $i \in \{n/2, \dots, n-1\}$  and this gives immediately a recursive algorithm (*i.e.* compute  $P_0, P_1, R_0, R_1$  and reduce the problem to the multipoint evaluation of  $R_0$  and  $R_1$  of degree  $n/2 = 2^{k-1}$ ).

Let  $A_i(x) = (x - \alpha_i)$  for  $i \in \{0, \dots, n-1\}$  and  $P_{i,j} = A_{j2^i} A_{j2^i+1} \dots A_{j2^i+2^{k-i}-1}$  for  $i \in \{0, \dots, k\}$  and  $0 \leq j < 2^{k-i}$ . We have  $P_{0,j} = A_j$  and  $P_{i+1,j} = P_{i,2j} P_{i,2j+1}$  so for  $i \in \{0, \dots, k\}$  we can compute recursively all polynomials  $P_{i,j}$  and  $0 \leq j < 2^{k-i}$  in  $2^{k-i-1} O(M(2^i)) = O(M(n))$  operations in  $\mathbb{Z}_N$  where  $M(i)$  denotes the arithmetic complexity to compute the product of two polynomials of degree  $i$  in  $\mathbb{Z}_N[x]$ . Overall, the computation of all polynomials  $P_{i,j}$  requires  $O(M(n) \log n)$  operations in  $\mathbb{Z}_N$  using a tree.

The polynomials  $R_0$  and  $R_1$  can be computed using  $O(M(n))$  operations in  $\mathbb{Z}_N$  (using a Newton inversion), hence the complexity  $T(n)$  of the recursive algorithm satisfies  $T(n) = 2T(n/2) + O(M(n))$  and therefore  $T(n) = O(M(n) \log n)$ .

The multipoint evaluation of univariate polynomials has found numerous application in cryptanalysis (*e.g.* [12,9]). In our case, it is clear that using this technique will lead to retrieve the seed. For example, using the same notations as in preliminaries, suppose that we know the value of  $v_i$  and want to retrieve the value of  $x_{i-1}$  of the generator. That can be done by multipoint evaluating the polynomial of degree  $e(2^{r/2} + 1)$ :

$$P(X) = (X^e - v_i)((X + 1)^e - v_i)((X + 2)^e - v_i)\dots((X + 2^{r/2})^e - v_i) \mod N$$

on the points  $k \cdot 2^{r/2}$  for  $k = 0, \dots, 2^{r/2}$  in order to find  $k_c$  such that  $P(k_c \cdot 2^{r/2}) = 0 \mod N$ . Then, one searches the value of  $x_{i-1}$  on the form  $k_c \cdot 2^{r/2} + \ell$  for  $\ell = 0, \dots, 2^{r/2}$ . This technique requires  $\tilde{O}(e \cdot 2^{r/2})$  operations in  $\mathbb{Z}_N$ . Its complexity is linear in  $e$  but, as mentioned above,  $e$  is chosen as small as possible in practice. Moreover, one has to store the first tree, i.e  $2^{r/2}$  polynomials.

*Remark 3.* This algorithm can be applied to attack the RSA encryption system when used to encrypt a short secret key of a symmetric cipher. Our algorithm is slightly less efficient than the one in [7] but it always succeeds (whereas recovering a 40-bit plaintext for instance is successful only with probability 0.39 in [7]).

## 4.2 Coppersmith's Method

Another technique is based on the well-known Coppersmith's method for the case of a modular univariate polynomial. In 1996, Coppersmith introduced lattice-based techniques for finding small roots on univariate and bivariate polynomial equations in polynomial time [11,10]. Some recalls are done in the full version. In our case, starting from the equation  $x_{i-1}^e = v_i \mod N$ , we can define the following modular univariate polynomial  $f$  as  $f(x) = x^e - v_i \mod N$ . The value  $x_{i-1}$  represents a small modular root of this polynomial. However, our root of size  $r$  is not enough small for this technique which requires the root to be less than  $N^{1/e}$ , i.e  $r < n/e$  (see [11]). To circumvent this problem, one can guess  $j$  bits of  $x$  in order to have  $r - j < n/e$  and then apply Coppersmith's method for each guess. Instead of  $f$ , one uses the polynomial  $g$  of degree  $e$ :

$$g(x) = (\lambda + x)^e - v_i \mod N$$

with  $\lambda$  the guessed value of  $j$  bits. The truncated value of  $x_{i-1}$  denoted by  $x_{i-1}^{tr}$  is a small modular root of  $g$ . Its degree being the same, the asymptotic condition on the size of the root remains the same. The following theorem, proved in the full version, establishes the condition on the bound:

**Theorem 5.** *Using the set of polynomials  $\{x^j g^i | j \leq e-1 \wedge i \leq p-1\} \cup \{g^p\}$  with  $p \in \mathbb{N}$ , Coppersmith's method will return  $x_{i-1}^{tr}$  as long as  $x_{i-1}^{tr} < N^\delta$  with:*

$$\delta = \frac{\sum_{i=1}^p (e-1)(i-1) + i}{\sum_{i=1}^{ep} i} = \frac{ep - e + 2}{e^2 p + 2}$$

Starting from a basis  $(b_1, \dots, b_w)$  of a lattice of  $\mathbb{Z}^m$ , this technique works in complexity  $O(w^4 m \log B(w + \log B))$  with  $B = \max_{1 \leq i \leq w} \|b_i\|$ , and we have to store the lattice of size  $w + m$ . By denoting  $x_{i-1}^{tr} < N^{\frac{1}{e} \cdot (1-\epsilon)}$  with  $0 < \epsilon < 1$ , one can determine the minimal value for  $p$  in order to retrieve the root, i.e  $p = \frac{e-1-\epsilon}{e\epsilon} = O(1/\epsilon)$ . The number of polynomials  $w$  being equal to  $e(p-1) + 1 = O(\frac{e}{\epsilon})$ , those of monomials  $m$  being equal to  $ep + 1 = O(\frac{e}{\epsilon})$  and  $\log B = n$ , this technique will require  $O(2^{r - \frac{n}{e} \cdot (1+\epsilon)} (\frac{e^3 n}{\epsilon^6} + \frac{e^5 n^2}{\epsilon^5}))$  in time and  $O(\frac{e}{\epsilon})^2$  in memory.

## 5 Conclusion

In this paper, for the first time, we have shown that, for all recommended parameters, we are able to predict the Micali-Schnorr pseudorandom generator faster than by an exhaustive search by using time/memory tradeoff or time/memory/data tradeoff attacks. These attacks are feasible only because of the specificity of this generator that uses only a small number to iterate and it remains a open problem to design a time/memory tradeoff algorithm able to infer sequences produced by the BBS or the RSA generator (in the range of parameters not covered by Herrmann and May techniques [20]).

We have also proposed three techniques (the last one is explained in the full version) to reverse the generator and retrieve the generator seed. An interesting open question is to decrease the memory requirement of our algorithms. The  $\rho$  or  $\lambda$  methods for factoring and discrete logarithms (which were invented by Pollard) use pseudorandom walks and require polynomial (or even constant) memory rather than exponential as in our time/memory tradeoffs. They can be applied to attack Gennaro's efficient pseudorandom generator based on the discrete logarithm [15] but it remains open to adapt this approach to predict the Micali-Schnorr pseudorandom generator.

**Acknowledgements.** This work was supported in part by the French ANR-12-JS02-0004 ROMANTIC Project.

## References

1. Alexi, W., Chor, B., Goldreich, O., Schnorr, C.-P.: RSA and Rabin functions: Certain parts are as hard as the whole. *SIAM J. Comput.* 17(2), 194–209 (1988)
2. Babbage, S.: A space/time tradeoff in exhaustive search attacks on stream ciphers. *IEE Conference Publication - European Convention on Security and Detection*, vol. 408 (1995)
3. Barkan, E., Biham, E., Shamir, A.: Rigorous bounds on cryptanalytic time/Memory tradeoffs. In: Dwork, C. (ed.) *CRYPTO 2006*. LNCS, vol. 4117, pp. 1–21. Springer, Heidelberg (2006)
4. Biryukov, A., Shamir, A.: Cryptanalytic time/memory/data tradeoffs for stream ciphers. In: Okamoto, T. (ed.) *ASIACRYPT 2000*. LNCS, vol. 1976, pp. 1–13. Springer, Heidelberg (2000)
5. Blum, L., Blum, M., Shub, M.: A simple unpredictable pseudo-random number generator. *SIAM J. Comput.* 15(2), 364–383 (1986)

6. Blum, M., Micali, S.: How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.* 13(4), 850–864 (1984)
7. Boneh, D., Joux, A., Nguyn, P.Q.: Why textbook ElGamal and RSA encryption are insecure. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 30–43. Springer, Heidelberg (2000)
8. Bostan, A., Gaudry, P., Schost, . Linear recurrences with polynomial coefficients and application to integer factorization and Cartier-Manin operator. *SIAM J. Comput.* 36(6), 1777–1806 (2007)
9. Chen, Y., Nguyn, P.Q.: Faster algorithms for approximate common divisors: Breaking fully-homomorphic-encryption challenges over the integers. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 502–519. Springer, Heidelberg (2012)
10. Coppersmith, D.: Finding a small root of a bivariate integer equation; factoring with high bits known. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 178–189. Springer, Heidelberg (1996)
11. Coppersmith, D.: Finding a small root of a univariate modular equation. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 155–165. Springer, Heidelberg (1996)
12. Coron, J.-S., Joux, A., Mandal, A., Naccache, D., Tibouchi, M.: Cryptanalysis of the RSA subgroup assumption from TCC 2005. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 147–155. Springer, Heidelberg (2011)
13. Fiat, A., Naor, M.: Rigorous time/space trade-offs for inverting functions. *SIAM J. Comput.* 29(3), 790–803 (1999)
14. Fiduccia, C.: Polynomial evaluation via the division algorithm: The Fast Fourier Transform revisited. In: Fischer, P., Zeiger, H.P., Ullman, J., Rosenberg, A. (eds.) 4th Annual ACM Symposium on Theory of Computing, pp. 88–93. ACM (1972)
15. Gennaro, R.: An improved pseudo-random generator based on the discrete logarithm problem. *Journal of Cryptology* 18(2), 91–110 (2005)
16. Goli, J.D.: Cryptanalysis of alleged A5 stream cipher. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 239–255. Springer, Heidelberg (1997)
17. Hstad, J., Impagliazzo, R., Levin, L.A., Luby, M.: A pseudorandom generator from any one-way function. *SIAM J. Comput.* 28(4), 1364–1396 (1999)
18. Hstad, J., Nslund, M.: The security of all RSA and discrete log bits. *J. ACM* 51(2), 187–230 (2004)
19. Hellman, M.E.: A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory* 26(4), 401–406 (1980)
20. Herrmann, M., May, A.: Attacking power generators using unravelled linearization: When do we output too much? In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 487–504. Springer, Heidelberg (2009)
21. Micali, S., Schnorr, C.-P.: Efficient, perfect polynomial random number generators. *Journal of Cryptology* 3(3), 157–172 (1991)
22. Oechslin, P.: Making a faster cryptanalytic time-memory trade-off. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 617–630. Springer, Heidelberg (2003)
23. Schroepel, R., Shamir, A.: A  $T = O(2^{n/2})$ ,  $S = O(2^{n/4})$  algorithm for certain NP-complete problems. *SIAM J. Comput.* 10(3), 456–464 (1981)
24. Steinfeld, R., Pieprzyk, J., Wang, H.: On the provable security of an efficient RSA-based pseudorandom generator. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 194–209. Springer, Heidelberg (2006)

# An Improved Algorithm for Extraction of Exact Boundaries and Boundaries Inclusion Relationship

Tao Hu<sup>1,2</sup>, Xianyi Ren<sup>1,2</sup>, and Jihong Zhang<sup>1,2</sup>

<sup>1</sup> Institute of Information Technology, Shenzhen Institute of Information Technology, 518171, Shenzhen, China  
happy.hut@163.com

<sup>2</sup> Shenzhen Key Laboratory of Visual Media Processing and Transmission, 518171, Shenzhen, China

**Abstract.** Boundary extraction algorithm proposed by Capson can get the same or even better performance as the commercial software such as *VisionPro* and *Halcon*. Unfortunately, the algorithm cannot extract the inclusion relationship between boundaries, which greatly reduce its attractiveness. Two improvements is proposed to improve the original algorithm in this paper, one is to improve the precision of boundaries by splitting points and merging points, another one is to design new data-structure and rules to implement acquiring deep inclusion relationship between external and internal boundaries. Experimental results show that the improvements increase a little consuming time but make the original algorithm more attractive.

**Keywords:** boundary extraction, inclusion relationship, run-length, splitting point, merging point, pattern recognition.

## 1 Introduction

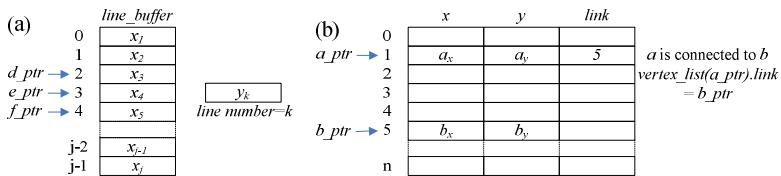
Boundary extraction is a basic and important topic in image processing[1,2], many researchers have devoted to it and many algorithms have been present. The conventional boundary extraction algorithms can be classified into three categories. 1) The first one is border following method proposed systematically by Rosenfeld in 1970[3], this algorithm is popular for its simplicity, and is improved by many researchers [4-9], but it has many drawbacks, foremost is the requirement to store the whole image data which obviously increases the memory requirement. Furthermore, it involves the neighborhood operation which increases the complexity of memory management and the amount of computation. 2) The second category of algorithms uses a window sliding the image data in a raster scanning fashion, by observing the sequence of patterns in the window, boundaries are extracted [10,11]. This category of algorithms avoids storing the whole image data for its sequential visiting way, instead storing no more than two lines data at a time. But it still has the drawbacks of neighborhood operation caused by the sliding window. 3) The third category of algorithms firstly implements run-length encoding through the raster scanning, and extract

boundary by analyzing the connectivity between the run-lengths of two adjacent lines [12-15]. This category of algorithms decreases the complexity further by avoiding the neighborhood operation in the above two categories of algorithms, but there are limitations more or less to the existing algorithms of this category. The algorithms proposed by Pavlidis, Kim and Quek all needs two-pass processing to finish the boundary extraction, which causes them fail to extract the boundary “on the fly”, so they does not satisfy the requirement of real-time extraction. The algorithm proposed by Capson can perform the boundary extraction “on the fly”, but distortion occurs at the concave portion of the boundaries, it does not satisfy the requirement of precision. Furthermore, the extraction of the deep inclusion relationship between external boundaries and internal boundaries is necessary, but most of the existing algorithms do not realize it.

“Defects cannot belittle virtues”, Capson’s boundary extraction algorithm can get the same or even better performance as the commercial software such as *Halcon* and *VisionPro*, which will be shown in the following “experiments and result” section. This paper is presented to address how to improve Capson’s algorithm to extract the exact boundaries with deep inclusion relationship.

## 2 Improved Boundary Extraction Algorithm

### 2.1 Outline of Capson’s Algorithm



**Fig. 1.** (a) *line\_buffer* data structure; (b)*Vertex\_list* data structure (Reproduced from [9])

Capson’s algorithm operates directly on segments which are defined by each pair of successive transition points yielded from RLE data. A part of corresponding edge points of segments held in the *line\_buffer* structure as Fig.1 (a) shows turn into vertex points and are added to the *vertex\_list* data structure as Fig.1 (b) shows, and the connectivity of segments in consecutive lines decides the connection of these vertex points and the creation, split, merging and termination of objects which are organized by the *active\_list* and the *complete\_list*. All boundaries are extracted with a single-pass of the image data in raster-scanned way. Each object in the *complete\_list* indicates an external boundary and holds the pointer to the starting vertex point of the external boundary; each hole in the *complete\_list* indicates an internal boundary and holds the pointer to the starting vertex point of the internal boundary. External boundaries are represented by counterclockwise linked vertex points and internal boundaries by clockwise linked vertex points.

## 2.2 Unsettled Questions in Capson's Algorithm and the Corresponding Solutions

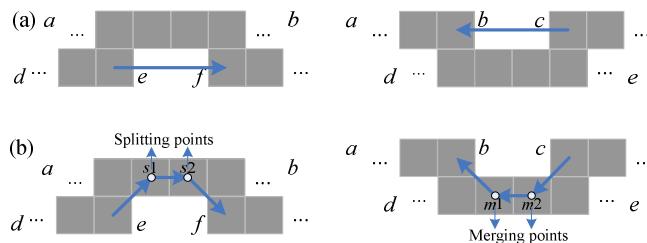
There are two unsettled questions in Capson's algorithm, one is the distortion that occurs in concave portion, and another is that it does not address how to extract the deep inclusion relationship between external and internal boundaries in his paper.

### 2.2.1 Solution to the Improvement of Boundaries Precision

The first problem lies in the splitting and merging of segments, which can be solved by introducing the splitting points and merging points as Fig.2 shows, and the new connection way can be concluded as follows:

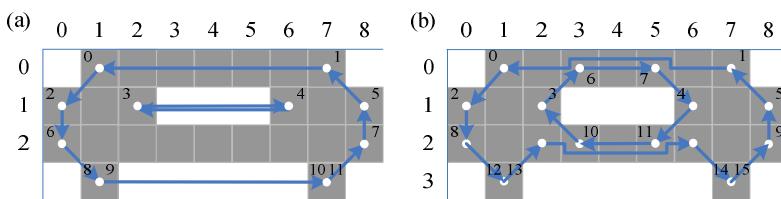
1) When segment splits, two splitting points  $s1$  and  $s2$  are added to the data structure. And connect  $e$  to  $s1$ ,  $s1$  to  $s2$ , and  $s2$  to  $f$ , instead of  $e$  to  $f$  directly.

2) When segments merge, two merging points  $m1$  and  $m2$  are added to the data structure, and connect  $c$  to  $m2$ ,  $m2$  to  $m1$ , and  $m1$  to  $c$ , instead of  $c$  to  $b$  directly.



**Fig. 2.** (a) The old connection way (b) The new connection way with the generating of splitting points and mergint points

A simple figure pattern but containing all the connection situations is used to show the difference between the origin algorithm and the improved one, as Fig.3 shows.



**Fig. 3.** (a) Boundaries extracted by the origin algorithm; (b) Boundaries extracted by the improve algorithm

### 2.2.2 Solutions to Building the Deep Inclusion Relationship of Holes and Objects

Two fields are added to *active\_list* data structure:

*parent\_hole* is a pointer used to indicate the inclusion relationship between an object and a hole when the object is included by the hole. Furthermore, this field of a hole is null generally.

*son\_object\_list* is a pointer list, when an object is included by a hole, the pointer to the object would be inserted to this list of the hole. Furthermore, this list of an object is null generally.

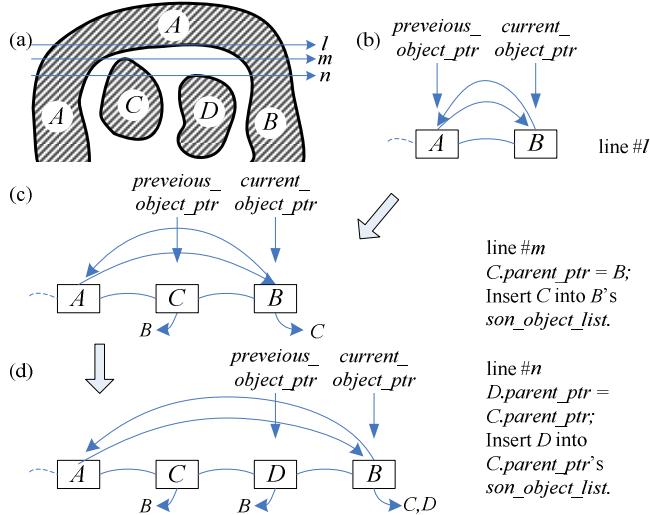
Moreover, the four operations on the *active\_list* all should be modified to build the deep inclusion relationship of holes and objects.

### 1. Updating for Creation of an Object Node

There are two cases to be considered in addition, suppose the object which *previous\_object\_ptr* currently points to is *pre\_obj*,

CASE 1 When *pre\_obj* has split, the new object to be created would be the son object of the object which *pre\_obj.forward* points to.

CASE 2 When *pre\_obj* owns a hole parent, which means *pre\_obj.parent\_hole* does not point to null, the new object to be created would be the son object of the object which *pre\_obj.parent* points to.



**Fig. 4.** (a) example objects; (b) *active\_list* update for (a) after line #l and before line #m; (c) *active\_list* update for (a) in line #m, C is created(CASE 1); (d) *active\_list* update for (a) in line #n, D is created(CASE 2)

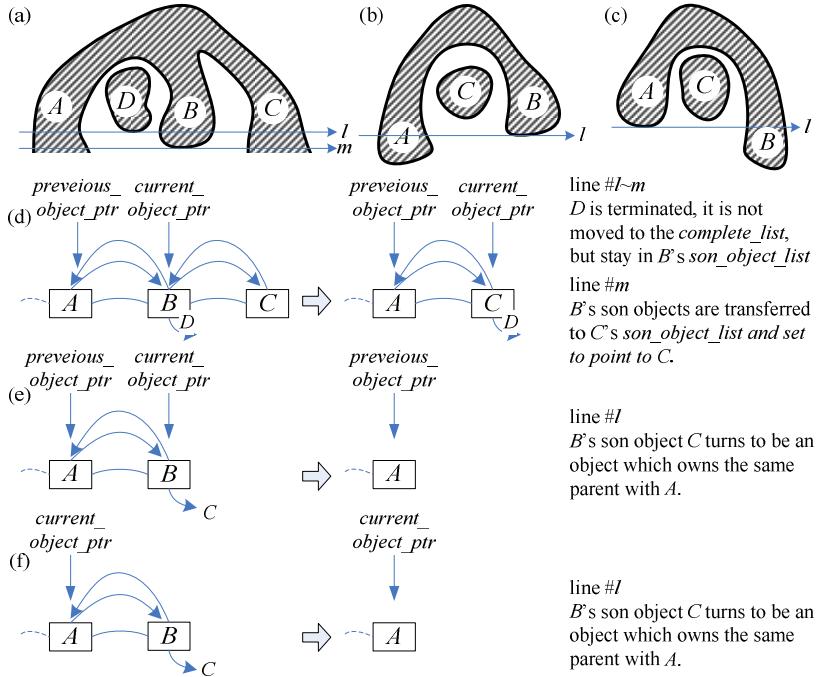
### 2. Updating for Termination of a Node

There are three cases to be considered:

CASE 1 When both the *forward* and *back* fields do not point to null, the son objects will be transferred to the *son\_object\_list* of the node which *forward* points to.

CASE 2 When the *back* field does not point to null, but the *forward* field does, the son objects will turn to be the son objects of the node which *parent\_hole* points to.

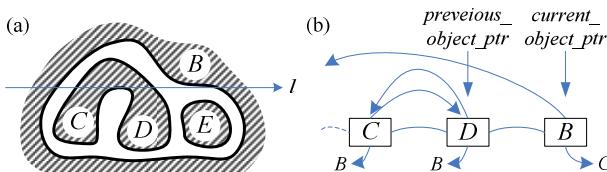
CASE 3 When the *forward* field does not point to null, but the *back* field does, the son objects owned by the node which *forward* points to will turn to be the son objects of the node which *parent\_hole* points to.



**Fig. 5.** (a)(b)(c) example objects; (d) *active\_list* update for (a)(CASE 1); (e) *active\_list* update for (b)(CASE 2); (f) *active\_list* update for (c)(CASE 3)

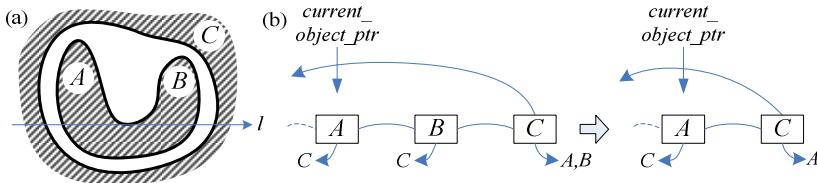
### 3. Updating for Split

When an object region splits, a node is inserted into the *active\_list* immediately to the right of the current node, besides, *parent\_hole* will point to the same parent with the current node, which makes the possible following new object inherit the same parent. It is worth noting that the new inserted node needs not be added to its parent's *son\_object\_list*.



**Fig. 6.** (a) example object; (b) *active\_list* in line #*l*

#### 4. Updating for Merge



**Fig. 7.** (a) example objects; (b) *active\_list* update from before line#*l* and after line# *l*

When a node and its immediate right neighbor in the *active\_list* are found merging, the immediate right neighbor's *parent\_hole* does not point to null, it will be deleted from the *active\_list*, the corresponding pointer will be removed from its parent's *son\_object\_list*.

Need to mention that there are some cases for merge not considered by original Capson's algorithm, which will result in failing to extract all boundaries of images with complex patterns[16]. These cases should be considered here by referring to [16], and the corresponding rules should be extended similarly to build the inclusion relationship.

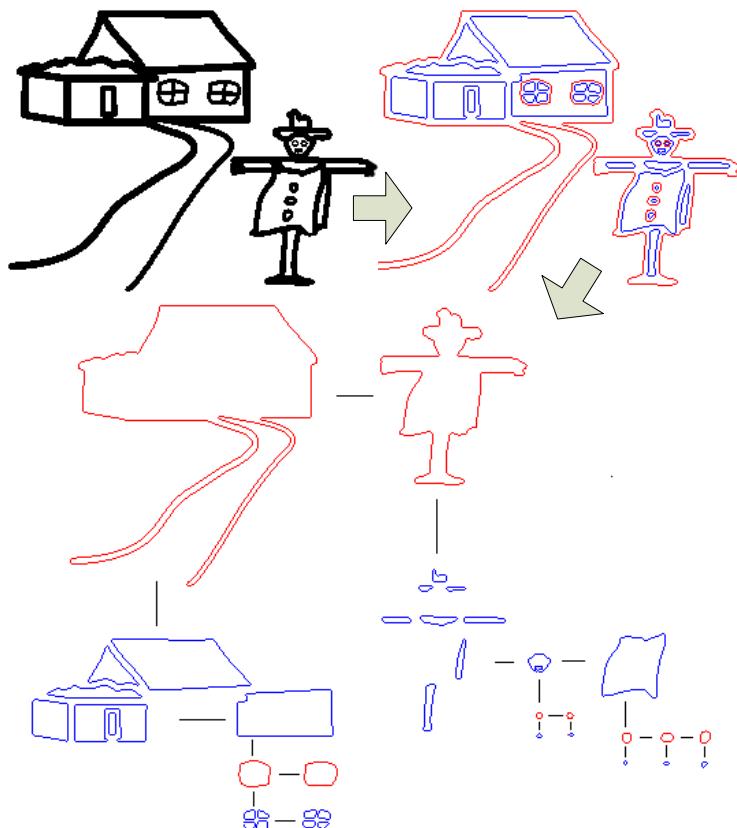
### 3 Experiments and Results

The improved algorithm is implemented with C++ programming language and tested in the Window XP on a PC with AMD Sempron™ 2600+ 1.60GHz and 512MB memory. Three sets of experiments are designed to show the performance of the improved algorithm in different aspects.

1) The first experiment: an example image containing huts and a scarecrow is used to test the improvement in the extraction of boundaries deep inclusion relationship, as Fig. 8 shows. And table 1 shows the performance comparison result between the improved algorithm and the original one. (The consuming-time is the total of 1000 times repeated operations.)

**Table 1.** Performance comparison between the origin algorithm and the improved algorithm

| Consuming-time                                              | t (ms/1000times) |
|-------------------------------------------------------------|------------------|
| Original algorithm<br>(without deep inclusion relationship) | 0.845            |
| Improved algorithm<br>(with deep inclusion relationship)    | 0.856            |

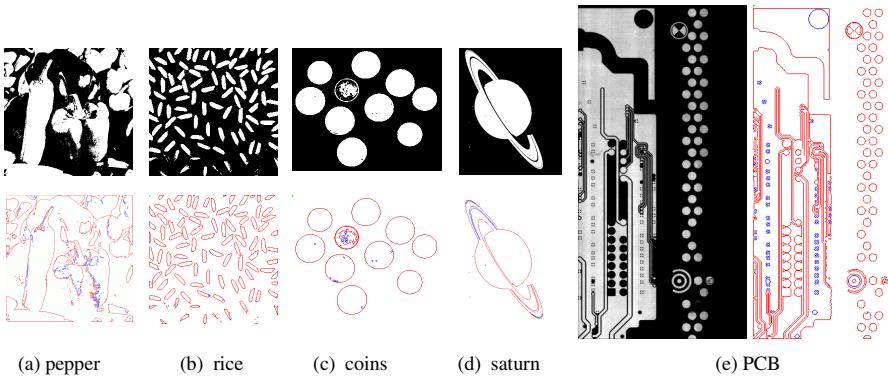


**Fig. 8.** Extraction of boundaries and deep inclusion relationship between external and internal boundaries, with red and blue color to distinguish them respectively

2) The second experiment: six images shown in Fig.9 are used to test the performance difference between classical boundary extraction algorithms and the improved one, the comparison result is shown in Table 2.

**Table 2.** Performance comparison between the improved algorithm and classical algorithms

| Image name                               | Image size | Ren[7]  | Chang[8] | Choy[11] | The improved |
|------------------------------------------|------------|---------|----------|----------|--------------|
| pepper                                   | 512×512    | 3.2ms   | 3.9ms    | 4.6ms    | 4.4ms        |
| Lenna                                    | 512×512    | 2.8ms   | 3.3ms    | 3.8ms    | 3.4ms        |
| rice                                     | 256×256    | 0.69ms  | 0.67ms   | 0.93ms   | 0.82ms       |
| coins                                    | 300×246    | 0.38ms  | 0.45ms   | 0.62ms   | 0.29ms       |
| saturn                                   | 1200×1500  | 6.3ms   | 10.3ms   | 13.6ms   | 3.7ms        |
| PCB                                      | 4096×8192  | 135.5ms | 204.7ms  | 227.6ms  | 83.1ms       |
| Can extract deep inclusion relationship? |            | No      | No       | Yes      | Yes          |



**Fig. 9.** Boundaries extraction with the improved algorithm for several images

3) The third experiment: the PCB image shown in Fig. 9.(e) with the size of 4096×8192 is used to test the performance difference between the improved algorithm and commercial software such as *Vision Pro* and *Halcon*, and the comparison result is shown in Table 3. (The improved algorithm is optimized by MMX/SSE instructions, so do *Vision Pro* and *Halcon*.)

**Table 3.** Performance comparison between the improved algorithm and commercial software

| Consuming-time(unit: ms) |       |       |       |       |       |         |
|--------------------------|-------|-------|-------|-------|-------|---------|
|                          | 1     | 2     | 3     | 4     | 5     | average |
| <i>Vision Pro</i>        | 79.3  | 83.6  | 85.9  | 83.2  | 84.2  | 83.24   |
| <i>Halcon</i>            | Step1 | 85.9  | 94.0  | 84.1  | 90.7  | 87.94   |
|                          | Step2 | 32.4  | 31.5  | 30.5  | 30.5  | 31.24   |
| Total                    | 118.3 | 125.5 | 114.6 | 121.2 | 117.2 | 119.18  |
| The improved             | 83.1  | 84.6  | 82.8  | 82.7  | 84.8  | 83.60   |

## 4 Conclusions

From the results shown in table 1~3, we can find that the improved algorithm 1) can extract exact boundaries and deep boundaries inclusion relationship; 2) can get the same or even better performance as the commercial software such as *VisionPro* and *Halcon*; 3) costs a little more than the original one, but make it more attractive.

**Acknowledgement.** This work is supported by the National Natural Science Foundation of China (61271420), the Natural Science Foundation of Guangdong Province, China (S2011040000662, S2011010006115), and Natural Science Foundation of Shenzhen Institute of Information Technology, China(YB201001).

## References

1. Mitraphont, J.L., Limkonglap, U.: Using Contour Analysis to Improve Feature Extraction in Thai Handwritten Character Recognition Systems. In: 7th IEEE International Conference on Computer and Information Technology, pp. 668–673. IEEE Computer Society, USA (2007)
2. Kruatrachue, B., Moongfangklang, N., Siriboon, K.: Fast Document Segmentation Using Contour and X-Y Cut Technique. In: Proceedings of World Academy of Science, Engineering and Technology, vol. 5, pp. 27–29. World Enformatika Society, Turkey (2005)
3. Rosenfeld, A.: Connectivity in Digital Pictures. *J. ACM* 17(1), 146–160 (1970)
4. Pavlidis, T.: Algorithms for Graphics and Image Processing, pp. 142–148. Computer Science Press, Rockville(America) (1982)
5. Wu, L.-D., Lin, Y.-Q.: Crack Based Contour Tracing and Tree Structure of Contours. *Chinese J. Computers* 19(6), 457–465 (1996) (Chinese)
6. Liu, X.-B., Xiang, J.-C., Xie, L.-H.: An Improved Contour Tracing Algorithm. *Computer Engineering and Application* 29(1), 61–63 (2005) (Chinese)
7. Ren, M.W., Yang, J.Y., Sun, H.: Tracing Boundary Contours in a Binary Image. *Image and Vision Computing* 20(2), 125–131 (2002)
8. Chang, F., Chen, C.J., Lu, C.J.: A Linear-Time Component-Labeling Algorithm Using Contour Tracing Technique. *Computer Vision and Image Understanding*, 206–220 (2004)
9. Wagenknecht, G.: A Contour Tracing and Coding Algorithm for Generating 2D Contour Codes from 3D Classified Objects. *Pattern Recognition* 40(4), 1294–1306 (2007)
10. Lunscher, W.H.H.J., Beddoes, M.P.: Fast Binary-Image Boundary Extraction. *Computer Vision, Graphics, and Image Processing* 38(3), 229–257 (1987)
11. Choy, C.T., Siu, W.C.: Single Pass Algorithm for the Generation of Chain-Coded Contours and Contours Inclusion Relationship. In: IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, pp. 256–259 (1993)
12. Pavlidis, T.: A Minimum Storage Boundary Tracing Algorithm and Its Application to Automatic Inspection. *Systems, Man and Cybernetics* 8(1), 66–69 (1978)
13. Kim, S.-D., Lee, J.-H., Kim, J.-K.: A New Chain-Coding Algorithm for Binary Images Using Run-Length Codes. *Computer Vision, Graphics, and Image Processing* 41(1), 114–128 (1988)
14. Quek, F.K.H.: An Algorithm for the Rapid Computation of Boundaries of Run-Length Encoded Regions. *Pattern Recognition* 33(10), 1637–1649 (2000)
15. Capson, D.W.: An Improved Algorithm for the Sequential Extraction of Boundaries from a Raster Scan. *Computer Vision, Graphics, and Image Processing* 28(1), 109–125 (1984)
16. Hu, T., Guo, B.-P., Guo, X.: An Improved Run-Based Boundary Extraction Algorithm. *Journal of Shenzhen University Science and Engineering* 26(4), 405–410 (2009)

# Straight-Line Monotone Grid Drawings of Series-Parallel Graphs

Md. Iqbal Hossain and Md. Saidur Rahman

Graph Drawing and Information Visualization Laboratory,  
Department of Computer Science and Engineering,  
Bangladesh University of Engineering and Technology  
`{mdiqbalhossain, saidurrahman}@cse.buet.ac.bd`

**Abstract.** A monotone drawing of a graph  $G$  is a straight line drawing of  $G$  where a monotone path exists between every pair of vertices of  $G$  in some direction. Recently monotone drawings of graphs have been discovered as a new standard for visualizing graphs. In this paper we study monotone drawings of series-parallel graphs in a variable embedding setting. We show that a series-parallel graph of  $n$  vertices has a straight-line planar monotone drawing on a grid of size  $O(n) \times O(n^2)$ .

## 1 Introduction

A path  $P$  in a straight-line drawing of a graph is *monotone* if there exists a line  $l$  such that the orthogonal projections of the vertices of  $P$  on  $l$  appear along  $l$  in the order induced by  $P$ . A straight-line drawing of a graph is monotone if it contains at least one monotone path for each pair of vertices.

*Upward drawings* [4,8] are related to monotone drawings where every directed path is monotone with respect to vertical lines, while in a monotone drawing each monotone path, in general, is monotone with respect to a different line. Arkin *et al.* [3] showed that any strictly convex drawing of a planar graph is monotone and they gave an  $O(n \log n)$  time algorithm for finding such a path from  $s$  to  $t$ . The authors in [1] showed that every biconnected planar graph has a straight-line monotone drawing in real coordinate space. Angelini *et al.* [1] showed that every tree admits a straight-line planar monotone drawing in  $O(n) \times O(n^2)$  or  $O(n^{1.6}) \times O(n^{1.6})$  area. Every connected plane graph admits a monotone grid drawing on an  $O(n) \times O(n^2)$  grid using at most two bends per edges and an outerplane graph of  $n$  vertices admits a straight-line monotone drawing on a grid of area  $O(n) \times O(n^2)$  [2]. It is also known that not every plane graph (with fixed embedding) admits a straight-line monotone drawing [1].

So the natural question is whether every connected planar graph has a straight-line monotone drawing and what is the minimum area requirement for such a drawing on a grid. In this paper, we investigate this problem for a non-trivial subclass of planar graphs called “series-parallel graphs”. We show that every series-parallel graph admits a straight-line monotone drawing on an  $O(n) \times O(n^2)$  grid which can be computed in  $O(n \log n)$  time.

We now give an outline of our algorithm for constructing a monotone drawing of a series-parallel graph  $G$ . We construct an ordered “ $SPQ$ -tree” of  $G$ . We then assign a slope to each node of the  $SPQ$ -tree. We finally draw  $G$  on a grid taking into consideration the slope assigned to each node of the  $SPQ$ -tree.

The rest of the paper is organized as follows. Section 2 describes some of the definitions that we have used in our paper. Section 3 deals with straight-line monotone drawings of series-parallel graphs. Finally, Section 4 concludes our paper with discussions.

## 2 Preliminaries

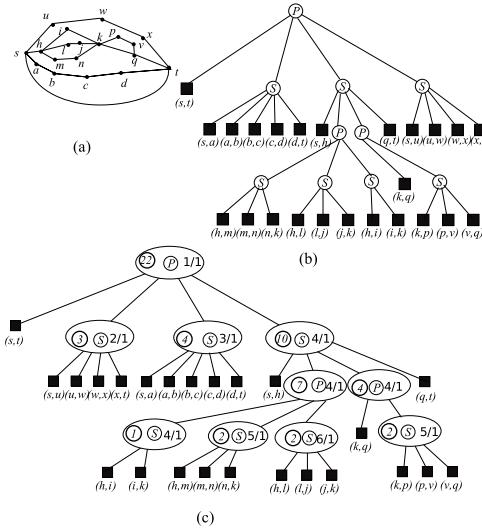
Let  $G = (V, E)$  be a connected graph with vertex set  $V$  and edge set  $E$ . A graph is *planar* if it can be embedded in the plane without edge crossings except at the vertices where the edges are incident. A *plane graph* is a planar graph with a fixed planar embedding. A plane graph divides the plane into some connected regions called *faces*. The unbounded region is called *outer face* and all the other faces are called *inner faces*. The vertices on the outer face are called *outer vertices* and all the other vertices are called *inner vertices*. A *cut vertex* is any vertex whose removal disconnects  $G$ . A *biconnected component*  $G'$  is a maximal biconnected subgraph of  $G$ .

A graph  $G = (V, E)$  is called a *series-parallel* graph (with source  $s$  and sink  $t$ ) if either  $G$  consists of a pair of vertices connected by a single edge or there exist two series-parallel graphs  $G_i = (V_i, E_i)$ ,  $i = 1, 2$ , with source  $s_i$  and sink  $t_i$  such that  $V = V_1 \cup V_2$ ,  $E = E_1 \cup E_2$ , and either  $s = s_1, t_1 = s_2$  and  $t = t_2$  or  $s = s_1 = s_2$  and  $t = t_1 = t_2$  [7]. A biconnected component of a series-parallel graph is also a series-parallel graph. By definition, a series-parallel graph  $G$  is a connected planar graph and  $G$  has exactly one source  $s$  and exactly one sink  $t$ .

**Fact 1.** *Let  $G = (V, E)$  be a series-parallel graph with the source vertex  $s$  and the sink vertex  $t$ . Assume that  $(s, t) \notin E(G)$ . Then  $G' = (V, E \cup (s, t))$  is a planar graph.*

A pair  $u, v$  of vertices of a connected graph  $G$  is a split pair if there exist two subgraphs  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$  satisfying the following two conditions: 1.  $V = V_1 \cup V_2$ ,  $V_1 \cap V_2 = \{u, v\}$ ; and 2.  $E = E_1 \cup E_2$ ,  $E_1 \cap E_2 = \emptyset$ ,  $|E_1| \geq 1$ ,  $|E_2| \geq 1$ . Thus every pair of adjacent vertices is a split pair. A *split component* of a split pair  $u, v$  is either an edge  $(u, v)$  or a maximal connected subgraph  $H$  of  $G$  such that  $u, v$  is not a split pair of  $H$ .

Let  $G$  be a biconnected series-parallel graph. Let  $(u, v)$  be an outer edge of  $G$ . The  $SPQ$ -tree [6,5]  $\mathcal{T}$  of  $G$  with respect to a reference edge  $e = (u, v)$  describes a recursive decomposition of  $G$  induced by its split pairs. Tree  $\mathcal{T}$  is a rooted ordered tree whose nodes are of three types:  $S$ ,  $P$  and  $Q$ . Each node  $x$  of  $\mathcal{T}$  corresponds to a subgraph of  $G$ , called its pertinent graph  $G(x)$ . Each node  $x$  of  $\mathcal{T}$  has an associated biconnected multigraph, called the *skeleton* of  $x$  and denoted by  $skeleton(x)$ . Tree  $\mathcal{T}$  is recursively defined as follows.



**Fig. 1.** (a) A series-parallel graph  $G$ , (b) an SPQ-tree  $\mathcal{T}$  of  $G$ , and (c) an illustration for slope assignment in  $\mathcal{T}'$  where subtrees are sorted on the number of vertices in each subtree, and number of vertices, node type and assigned slope of each node are written inside the node

*Trivial Case:* In this case,  $G$  consists of exactly two parallel edges  $e$  and  $e'$  joining  $s$  and  $t$ .  $\mathcal{T}$  consists of a single  $Q$ -node  $x$ , and the skeleton of  $x$  is  $G$  itself. The pertinent graph  $G(x)$  consists of only the edge  $e'$ .

*Parallel Case:* In this case, the split pair  $u, v$  has three or more split components  $G_0, G_1, \dots, G_k, k \geq 2$ , and  $G_0$  consists of only a reference edge  $e = (u, v)$ . The root of  $\mathcal{T}$  is a  $P$ -node  $x$ . The  $\text{skeleton}(x)$  consists of  $k + 1$  parallel edges  $e_0, e_1, \dots, e_k$  joining  $s$  and  $t$ , where  $e_0 = e = (u, v)$  and  $e_i, 1 \leq i \leq k$ , corresponds to  $G_i$ . The pertinent graph  $G(x) = G_1 \cup G_2 \cup \dots \cup G_k$  is the union of  $G_1, G_2, \dots, G_k$ .

*Series Case:* In this case the split pair  $u, v$  has exactly two split components, and one of them consists of the reference edge  $e$ . One may assume that the other split component has cut-vertices  $c_1, c_2, \dots, c_{k-1}$ ,  $k \geq 2$ , that partition the component into its blocks  $G_1, G_2, \dots, G_k$  in this order from  $t$  to  $s$ . Then the root of  $\mathcal{T}$  is an  $S$ -node  $x$ . The skeleton of  $x$  is a cycle  $e_0, e_1, \dots, e_k$  where  $e_0 = e, c_0 = u, c_k = v$ , and  $e_i$  joins  $c_{i-1}$  and  $c_i, 1 \leq i \leq k$ . The pertinent graph  $G(x)$  of node  $x$  is the union of  $G_1, G_2, \dots, G_k$ . Figure 1 shows a series-parallel graph and its SPQ-tree decomposition.

Let  $\mathcal{T}$  be the SPQ-tree of a series-parallel graph  $G$  and let  $x$  be a node of  $\mathcal{T}$ . The pertinent graph of  $x$  is denoted by  $\text{pert}(x)$ . For an  $S$ -node  $x$ , we denote by  $n(x)$  the number of vertices in  $\text{pert}(x)$  excluding  $s$  and  $t$ . For a  $P$ -node  $x$ , we denote by  $n(x)$  the number of vertices in  $\text{pert}(x)$  including  $s$  and  $t$ . According to the SPQ-tree decomposition, a  $P$ -node can not be the parent of another  $P$ -node

and an  $S$ -node can not be the parent of another  $S$ -node. Throughout the paper, by drawing of a node  $x$  in  $\mathcal{T}$  we mean the drawing of  $\text{pert}(x)$  of  $G$ .

### Monotone Drawings

Let  $p$  be a point in the plane and  $l$  be a half-line starting at  $p$ . The slope of  $l$ , denoted by  $\text{slope}(l)$ , is the angle spanned by a counter-clockwise rotation that brings a horizontal half-line starting at  $p$  and directed towards increasing  $x$ - coordinates to coincide with  $l$ .

Let  $\Gamma$  be a drawing of a graph  $G$  and let  $(u, v)$  be an edge of  $G$ . The half-line starting at  $u$  and passing through  $v$ , denoted by  $d(u, v)$ , is the *direction* of  $(u, v)$ . The direction of an edge  $e$  is denoted by  $d(e)$  and slope of  $e$  is denoted by  $\text{slope}(e)$ .

Let  $P(u_1, u_q) = (u_1, u_2, \dots, u_q)$  be a path in a straight-line drawing of a graph. Let  $e_i$  be the edge from  $u_i$  to  $u_{i+1}$  ( $1 \leq i \leq q-1$ ). Let  $e_j$  and  $e_k$  be two edges of the path  $P(u_1, u_q)$  such that  $\text{slope}(e_j) \geq \text{slope}(e_i)$  and  $\text{slope}(e_k) \leq \text{slope}(e_i)$  for  $i = 1, \dots, q-1$ . We call  $e_j$  and  $e_k$  *extremal edges* of the path  $P(u_1, u_q)$ . The path  $P(u_1, u_q)$  is a *monotone path* with respect to a direction  $d$  if the orthogonal projections of vertices  $u_1, \dots, u_q$  on a line along the direction  $d$  appear in the same order as the vertices appear in the path.

Let  $P(u_1, u_q) = (u_1, u_2, \dots, u_q)$  be a monotone path. Let  $e_1$  and  $e_2$  be the extremal edges of  $P(u_1, u_q)$ . If we draw  $e_i$  at the origin of the axes,  $e_1$  and  $e_2$  create a closed wedge at the origin of the axes. The closed wedge delimited by  $d(e_1)$  and  $d(e_2)$  and containing all the half-lines  $d(u_i, u_{i+1})$ , for  $i = 1, \dots, q-1$ , is the *range* of  $P(u_1, u_q)$  and is denoted by  $\text{range}(P(u_1, u_q))$ , while the closed wedge delimited by  $d(e_1) - \pi$  and  $d(e_2) - \pi$ , and not containing  $d(e_1)$  and  $d(e_2)$ , is the *opposite range* of  $P(u_1, u_q)$  and is denoted by  $\text{opp}(P(u_1, u_q))$ .

We now recall some important properties of monotone paths from [1] as in the following two lemmas.

**Lemma 1.** *A path  $P(u_1, u_q) = (u_1, u_2, \dots, u_q)$  is monotone if and only if it contains two edges  $e_1$  and  $e_2$  such that the closed wedge centered at the origin of the axes, delimited by the two half-lines  $d(e_1)$  and  $d(e_2)$ , and has an angle smaller than  $\pi$ , contains all the half-lines  $d(u_i, u_{i+1})$ , for  $i = 1, \dots, q-1$ .*

**Lemma 2.** *Let  $P(u_1, u_q) = (u_1, \dots, u_q)$  and  $P(u_q, u_{q+k}) = (u_q, \dots, u_{q+k})$  be monotone paths. Then, path  $P(u_1, u_{q+k}) = (u_1, \dots, u_q, u_{q+1}, \dots, u_{q+k})$  is monotone if and only if  $\text{range}(P(u_1, u_q)) \cap \text{opp}(P(u_q, u_{q+k})) = \emptyset$ . Further, if  $P(u_1, u_{q+k})$  is monotone, then  $\text{range}(P(u_1, u_q)) \cup \text{range}(P(u_q, u_{q+k})) \subseteq \text{range}(P(u_1, u_{q+k}))$ .*

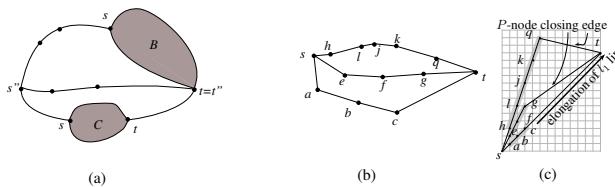
## 3 Monotone Grid Drawing

In this section we give an  $O(n \log n)$  time algorithm to find a straight-line planar monotone grid drawing of a series-parallel graph on an  $O(n) \times O(n^2)$  grid. To get a such drawing we first construct an ordered  $SPQ$ -tree. We then assign a slope to each node. We finally draw the graph on a grid taking into consideration the slopes assigned to each node of the tree. The details of our algorithm are as follows.

We assume that an edge exists between the source  $s$  and the sink  $t$  of the input series-parallel graph  $G$  otherwise we add a dummy edge between  $s$  and  $t$  of  $G$ . (Note that the graph remains planar after adding the dummy edge  $(s, t)$  by Fact 1.) Later we will show that the drawing of  $G$  obtained by our algorithm remains monotone even after removing the dummy edge  $(s, t)$  from the drawing. Clearly,  $G$  is a biconnected series-parallel graph (with the edge  $(s, t)$ ). Let  $\mathcal{T}$  be the  $SPQ$ -tree of  $G$  with respect to edge  $(s, t)$ . Then the root of  $\mathcal{T}$  is a  $Q$ -node  $r$  and the only child of  $r$  is a  $P$ -node  $x$ . We now re-root  $\mathcal{T}$  at  $x$ .

Let  $\mathcal{T}'$  be an ordered  $SPQ$ -tree obtained from  $\mathcal{T}$  as follows. We traverse each  $P$ -node of  $\mathcal{T}$ ; if any  $Q$ -node exists as a child of a  $P$ -node we put the  $Q$ -node as the leftmost child of the  $P$ -node. The rest of the children of the  $P$ -node are  $S$ -nodes and we draw them from left to right according to increasing order of the number of vertices in the subtree rooted at the respective  $S$ -node. We now assign a slope to each node of  $\mathcal{T}'$ . Let  $1/1, 2/1, \dots, (n-1)/1$  be  $n-1$  slopes in increasing order. Initially we assign the slope  $1/1$  to the root of  $\mathcal{T}'$ . We then traverse  $\mathcal{T}'$  to assign a slope to each node  $x$  of  $\mathcal{T}'$ . We first consider the case where  $x$  is a  $P$ -node. Let the slope assigned to  $x$  be  $\mu/1$ , and let  $x_1, x_2, \dots, x_k$  ( $k < n$ ) be the children of  $x$  in left to right order. We assign the slope  $\mu/1$  to the leftmost child  $x_1$ . We next assign the slope  $(\mu_{i_{max}}+1)/1$  to  $x_{i+1}$  where  $\mu_{i_{max}}/1$  is the largest slope assigned in the subtree rooted at  $x_i$ . We now consider the case where  $x$  is an  $S$ -node. Let the slope assigned to  $x$  be  $\mu/1$ , and let  $x_1, x_2, \dots, x_k$  ( $k < n$ ) be the children of  $x$  in left to right order. We assign the same slope  $\mu/1$  to  $x_i$  ( $i \leq k$ ). Thus the largest slope assigned to a vertex can be at most  $(n-1)/1$ .

We are now ready to draw  $G$  on a grid using the slope assigned to each node of  $\mathcal{T}'$ . Figure 1(c) illustrates the slope assignment to the nodes of the  $SPQ$ -tree for the graph  $G$  shown in Figure 1(a). Our algorithm uses a post-order traversal on the ordered  $SPQ$ -tree.



**Fig. 2.** (a) An example of a  $P$ -node for Cases 1 and 2, (b) a  $P$ -node  $x$ , and (c) a drawing of  $x$  on a grid

We first give a drawing algorithm for a  $P$ -node. Let  $x$  be a  $P$ -node with slope  $\mu/1$  assigned to it and let  $s$  and  $t$  be the source and the sink of  $x$ , respectively. Let  $x_1, x_2, \dots, x_k$  be the children of  $x$  in left to right order. Let  $\mu_1/1, \mu_2/1, \dots, \mu_k/1$  be the slopes assigned to  $x_1, x_2, \dots, x_k$ , respectively.

If  $x$  is not the root of  $\mathcal{T}'$ , let  $x'$  be the parent node of  $x$ , and let  $x''$  be the parent node of  $x'$ . Clearly  $x'$  is an  $S$ -node and  $x''$  is a  $P$ -node. Let  $\mu/1$  and  $\mu'/1$  be the slope assigned to  $x'$  and  $x''$ , respectively. Let  $s''$  and  $t''$  be the source and the sink of  $x''$ , respectively.

We denote the position of a vertex  $u$  by  $p(u)$ ;  $p(u)$  is expressed by its  $x$ - and  $y$ -coordinates as  $(p_x(u), p_y(u))$  on a grid. Let  $p(x)$  be a point on the grid. We place the source vertex of the  $P$ -node  $x$  on  $p(x)$ . Let  $A_x$  be a set of points where the neighbors of  $t$  in  $\text{pert}(x)$  are to be drawn.

We now have the following two cases to consider.

**Case 1:**  $t \neq t''$ . In this case (see the node labeled  $C$  in the Figure 2), we place  $s$  on  $p(x)$ . If  $x_1$  is a  $Q$ -node we leave it for now, and we draw the respective edge after placing the sink  $t$  of  $x$ . We add  $p(x)$  to  $A_x$ . Otherwise all  $x_i$  are  $S$ -nodes and we follow the drawing algorithm of  $S$ -node to draw each  $x_i$ .

After drawing all  $x_i$ , we elongate the  $l_1$  (the equation of  $l_1$  is  $y = \mu_1 \times x + p_x(x)$ ) line up to the point  $p(x_{end}) = (p_x(x) + n(x), p_y(x) + \mu_1 \times n(x))$  and place the sink  $t$  of  $x$  on  $p(x_{end})$ . Since  $n(x_1) \leq n(x_2) \leq n(x_3) \dots \leq n(x_k)$  and the slope  $\mu_1/1 < \mu_2/1 < \dots < \mu_k/1$ ,  $p(x_{i_{end}})$  is visible from  $p(x_{end})$ . We connect  $t$  to all points in  $A_x$  using straight line segments and we call each of these edges  *$P$ -node closing edge*. Note that the slope of edge  $(p(x_{i_{end}}), p(x_{end}))$  satisfies  $\pi/2 > \text{slope}(p(x_{i_{end}}), p(x_{end})) > -\pi/2$ .

**Case 2:**  $t = t''$ . Figure 2 illustrates an example of this case (see node  $B$ ).

Let  $p_x(Y'')$  be the largest  $x$ -coordinate used in the drawing of  $x''$ . If  $p_x(x) < p_x(Y'')$ , we set  $p(x) = (p_x(Y''), \frac{p_x(Y'') - p_x(x'')}{\mu})$ . We then place the  $s$  on  $p(x)$ . We now draw all the  $S$ -nodes according to the  $S$ -node drawing algorithm described later. Since the sink vertex of  $x$  and  $x''$  are same, we do not draw  $t$  in this step. All the end vertices of  $x_i$  will be connected at the drawing of the sink of  $x''$ . If  $x_1$  is a  $Q$ -node, we add  $p(x)$  in  $A_x$ .

We now describe an algorithm for drawing an  $S$ -node. Let  $x$  be an  $S$ -node of  $\mathcal{T}'$  with assigned slope  $\mu/1$ . Let  $x'$  be the parent node of  $x$  and let  $s'$  be the source vertex of  $x'$ . Let  $p(x')$  be the point where  $s'$  has already been placed. Clearly,  $x'$  is a  $P$ -node. Let  $l$  be a straight-line such that the equation of  $l$  is  $y = \mu/1 \times x + p_x(x')$ .

Assume first that all the children of  $x$  are  $Q$ -nodes. Then the  $\text{pert}(x)$  is a path. In this case we place the vertices of  $\text{pert}(x)$  on the line  $l$  sequentially on integer points. The last vertex of  $\text{pert}(x_i)$  lies on the point  $p_{end}(x) = ((p_x(x) + n(x), p_y(x) + \mu/1 \times n(x))$  ( $\text{slope}(l) = \mu/1$ ). Then we add the  $p_{end}(x)$  in  $A_{x'}$ . Assume now that some of the children of  $x$  are  $P$ -nodes. We traverse left to right subtrees of  $x$ . If  $x_i$  is a  $Q$ -node, we place corresponding vertices on the line  $l$ . If  $x_i$  is a  $P$ -node, we set  $p(x_i) = ((p_x(x) + i, p_y(x) + \mu/1 \times i)$  when the source vertex of  $x_i$  is not  $s'$ , otherwise  $p(x_i) = p(x')$ . We then use the drawing algorithm for  $P$ -nodes.

We now describe how we fix the coordinates for the drawing of a  $Q$ -node. Let  $x$  be a  $Q$ -node. The pertinent graph of  $x$  is an edge  $(u, v)$ . Note that  $u$  is already placed on the grid, since our drawing algorithm follows post-order traversal on the  $SPQ$ -tree. Let  $(\alpha, \beta)$  be the coordinate of  $u$ . If  $v$  is a sink of any  $P$ -node,

we handle this in the drawing of  $P$ -node closing edges. We thus assume that  $v$  is not a sink of any  $P$ -node. In this case we place vertex  $v$  at  $(\alpha + 1, \beta + \mu)$  on the line  $y = \mu x + \beta - \alpha\mu$ , where  $\mu/1$  is the slope assigned to  $x$ .

We call the algorithm described above Algorithm *Monotone-Draw*. We now have the following theorem.

**Theorem 1.** *Algorithm Monotone-Draw finds a monotone drawing of a series-parallel graph on a grid of size  $O(n) \times O(n^2)$  in  $O(n \log n)$  time.*

*Proof.* Let  $\Gamma$  be the drawing of  $G$  constructed by Algorithm Monotone-Draw. We now show that  $\Gamma$  is a monotone drawing of  $G$ . To prove the claim, we show that, a monotone path exists between every pair of vertices of  $G$  in  $\Gamma$ .

Let  $s$  and  $t$  be the source and the sink of  $G$ . In fact we will prove that a monotone path exists between every pair of vertices of  $G$  in the drawing of  $G \setminus (s, t)$  in  $\Gamma$ . Let  $v$  be a vertex in  $G$  such that  $v \notin \{s, t\}$ . We first show that there exist a monotone path between  $v$  and  $s$ , and between  $v$  and  $t$ . One can easily observe that a path  $P(s, v)$  exists such that no  $P$ -node closing edge is contained in  $P(s, v)$  and for each edge  $e \in P(v, s)$ ,  $\pi/2 > \text{slope}(e) \geq \pi/4$  holds. Then the path  $P(s, v)$  is monotone since the  $\text{range}(P(s, v))$  is smaller than  $\pi$ . On the other hand a path  $P(v, t)$  exists such that  $s \notin P(v, t)$ ,  $(s, t) \notin P(v, t)$  and  $P(v, t)$  may contain some  $P$ -node closing edges. The path  $P(v, t)$  is monotone since for each edge  $e \in P(v, t)$   $\pi/2 < \text{slope}(e) < -\pi/2$  holds. Similarly, any path  $P(s, t)$  in  $\Gamma \setminus (s, t)$  is monotone since for each edge  $e \in P(s, t)$   $\pi/2 < \text{slope}(e) < -\pi/2$  holds.

We now show that for every pair of vertices  $u, v \in G$  ( $v \notin \{s, t\}$ ,  $u \notin \{s, t\}$ ) there is a monotone path between  $u$  and  $v$  in  $\Gamma$ . Let  $P(u, s)$  and  $P(v, s)$  be two paths such that none of them contains a  $P$ -node closing edge and assume that  $e_1 = (u, u')$  lies on  $P(u, s)$  and  $e_2 = (v, v')$  lies on  $P(v, s)$ .

Let  $M$  and  $N$  be the two  $Q$ -nodes in  $\mathcal{T}'$  such that  $(u, u') \in \text{pert}(M)$  and  $(v, v') \in \text{pert}(N)$ , and let  $W$  be the lowest common ancestor of  $M$  and  $N$  in  $\mathcal{T}'$ . Let  $U$  and  $V$  be the children of  $W$  and ancestors of  $M$  and  $N$ , respectively. Let  $\mu_W, \mu_U, \mu_V, \mu_M, \mu_N$  be the slopes assigned to the nodes  $W, U, V, M$  and  $N$ , respectively. Since  $e_1$  and  $e_2$  are not  $P$ -node closing edges, the slopes of  $d(e_1)$  and  $d(e_2)$  are  $-\mu_M$  and  $-\mu_N$ , respectively.

We now have the following two cases to consider.

**Case 1:** *W is a P-node.* Without loss of generality we may consider  $\mu_M > \mu_N$ . So according to the slope assignment  $\mu_M \geq \mu_U > \mu_V \geq \mu_N$ . Let  $w$  and  $w'$  be the source and sink vertices of  $W$  in  $\Gamma$ . The path  $P(u, v)$  ( $w' \notin P(u, v)$ ) is composed of path  $P(u, w)$  and of path  $P(w, v)$ . Clearly, for each edge  $e \in P(u, w)$ , and  $e' \in P(w, v)$  it holds  $\mu_M \geq \text{slope}(e) \geq \mu_U$  and  $\mu_N \geq \text{slope}(e') \geq \mu_V$ , respectively. So we have  $\text{range}(P(u, w)) \cap \text{opp}(P(w, v)) = \emptyset$ . Then by Lemma 2,  $P(u, v)$  is a monotone path.

**Case 2:** *W is an S-node.* If  $M$  and  $N$  are children of the same  $S$ -node then the case is straight-forward, the path  $P(u, v)$  lies on a straight-line. Otherwise the path  $P(u, v)$  could have some  $P$ -node closing edge. let  $W'$  be the parent of  $W$ . Clearly  $W'$  is a  $P$ -node. Let  $w'$  be the source vertex of  $W'$  in  $\Gamma$ . Then the path  $P(u, v)$  ( $w' \notin P(u, v)$ ) is monotone with respect to a horizontal half-line.

Thus we have proved that  $\Gamma$  is a monotone drawing of  $G$ .

We are placing the sink of each  $P$ -node on the  $x = n(x)$  line. The drawings of all child nodes are inside the drawing of its parent  $P$ -node. So the largest  $x$ -coordinate of the drawing can be at most  $n$ . On the other hand we might get  $B$  type nodes (see Figure 2) recursively, and hence the  $y$ -coordinate can be up to  $O(n^2)$ . Thus the total grid size is  $O(n) \times O(n^2)$ .

We now analyze the required time for our algorithm. We construct  $SPQ$ -tree in linear-time, and  $O(n \log n)$  time is required to sort. We assign slopes to  $\mathcal{T}'$  in linear time. Thus the overall time complexity of the algorithm is  $O(n \log n)$ .  $\square$

## 4 Conclusion

In this paper we have studied monotone grid drawings of series-parallel graphs. We have shown that a series-parallel graph of  $n$  vertices has a straight-line planar monotone drawing on an  $O(n) \times O(n^2)$  grid and such a drawing can be found in  $O(n \log n)$  time. Finding straight-line monotone grid drawings of larger classes of planar graphs is remained as our future work.

**Acknowledgment.** We thank CodeCrafters International and Investortools, Inc. for supporting this research under the grant “CodeCrafters-Investortools Research Grant for CSE BUET”.

## References

1. Angelini, P., Colasante, E., Di Battista, G., Frati, F., Patrignani, M.: Monotone drawings of graphs. *Journal of Graph Algorithms and Applications* 16(1), 5–35 (2012)
2. Angelini, P., Didimo, W., Kobourov, S., Mchedlidze, T., Roselli, V., Symvonis, A., Wismath, S.: Monotone drawings of graphs with fixed embedding. In: Speckmann, B. (ed.) *GD 2011*. LNCS, vol. 7034, pp. 379–390. Springer, Heidelberg (2011)
3. Arkin, E.M., Connelly, R., Mitchell, J.S.: On monotone paths among obstacles with applications to planning assemblies. In: *Proceedings of the Fifth Annual Symposium on Computational Geometry*, SCG 1989, pp. 334–343. ACM, New York (1989)
4. Di Battista, G., Tamassia, R.: Algorithms for plane representations of acyclic digraphs. *Theoretical Computer Science* 61(2-3), 175–198 (1988)
5. Di Battista, G., Tamassia, R.: On-line maintenance of triconnected components with spqr-trees. *Algorithmica* 15(4), 302–318 (1996)
6. Di Battista, G., Tamassia, R., Vismara, L.: Output-sensitive reporting of disjoint paths. *Algorithmica* 23, 302–340 (1999)
7. Rahman, M.S., Egi, N., Nishizeki, T.: No-bend orthogonal drawings of series-parallel graphs. In: Healy, P., Nikolov, N.S. (eds.) *GD 2005*. LNCS, vol. 3843, pp. 409–420. Springer, Heidelberg (2006)
8. Samee, M.A.H., Rahman, M.S.: Upward planar drawings of series-parallel digraphs with maximum degree three. In: *WALCOM 2007*, pp. 28–45 (2007)

# Combination of Two-Machine Flow Shop Scheduling and Shortest Path Problems

Kameng Nip and Zhenbo Wang\*

Department of Mathematical Sciences, Tsinghua University, Beijing, 100084, China  
zwang@math.tsinghua.edu.cn

**Abstract.** This paper studies a combinatorial optimization problem which is obtained by combining the two-machine flow shop scheduling problem and the shortest path problem. The objective of the obtained problem is to select a subset of jobs constitutes a feasible solution to the shortest path problem, and to execute the selected jobs on two-machine flow shop to minimize the makespan. We argue that this problem is NP-hard, and propose two approximation algorithms with constant factor guarantee.

**Keywords:** two-machine flow shop scheduling, shortest path, combination of optimization problems, approximation algorithm.

## 1 Introduction

With the rapid development of science and technology, manufacturing, service and management are often integrated, and decision-makers have to deal with systems involve several characteristics from more than one well-known combinatorial optimization problems. To the best of our knowledge, few research have been done about the combination of optimization problems in literature.

Wang and Cui [10] first studied a combination of the parallel machine scheduling problem and the vertex cover problem. The goal is to select a subset of jobs that forms a vertex cover of a given graph and to execute these jobs on  $m$  identical parallel machines. They proposed an  $(3 - \frac{2}{m+1})$  - approximation algorithm for that problem. Wang et al. [11] have investigated a generalization of the above problem that combines the uniformly related parallel machine scheduling problem and a generalized covering problem. They proposed several approximation algorithms and mentioned as future research other combination of well-known combinatorial optimization problems. This is the core motivation for this work.

Let us consider the following scenario. We aim at building a railway between two specific cities. The railway needs to cross several adjacent cities, which is determined by a map (a graph). The processing time of manufacturing the rail track for each pair of cities is various. Manufacturing a rail track between two cities in the graph is associated with a job. The decision-maker needs to make two main decisions: (1) choosing a path to connect the two cities, and (2) deciding the

---

\* Corresponding author.

schedule of manufacturing the rail tracks on this path in the factory. In addition, the manufacturing of rail tracks follows several working stages, each stage must start after the completion of the preceding stages, and we assume that there is only one machine for each stage. We wish to accomplish the manufacturing as early as possible, i.e. minimize the completion time of the schedule. It is a standard flow shop scheduling problem. In this paper, we assume that there are only two stages, leading to the two-machine flow shop scheduling problem. How can a decision maker choose a feasible path such that the corresponding jobs can be manufactured as early as possible? This problem combines the structure of the two-machine flow shop scheduling problem and the shortest path problem.

Following the framework introduced by Wang et al. [11], we can see our problem as a combination of two optimization problems, two-machine flow shop scheduling and some shortest path problem. The former problem can be solved in  $O(n \log n)$  using Johnson's rule [6,8], and the classical shortest problem with non-negative edge weights can be solved in  $O(|V|^2)$  using Dijkstra's algorithm [1,5].

The contributions of this paper include: (1) the argument that the considered problem is NP-hard, and (2) two constant factor approximation algorithms.

The rest of the paper is organized as follows. In section 2, we give a formal definition of our problem, and then briefly review the two-machine flow shop scheduling problem and some shortest path problems. In section 3, we study the computational complexity of our problem. Section 4 provides two approximation algorithms for this problem and we conclude in section 5.

## 2 Preliminaries

### 2.1 Problem Description

We first introduce the following generalized shortest path problem.

**Definition 1.** *Given a directed graph  $G = (V, A, w^1, w^2)$  and two distinguished vertices  $s, t \in V$  with  $|A| = n$ . Each arc  $a_j \in A, j = 1, \dots, n$  is associated with two weights  $w_j^1, w_j^2$ , and we define the vector  $w^k = (w_1^k, w_2^k, \dots, w_n^k)$  for  $k = 1, 2$ . The goal of our shortest path problem is to find a directed path  $P$  from  $s$  to  $t$  to minimize  $f(w^1, w^2; x)$ , in which  $f$  is certain specific objective function and  $x \in \{0, 1\}^n$  is the decision variables such that  $x_j = 1$  if and only if  $a_j \in P$ .*

We denote  $SP$  instead of  $SP(G, s, t, f)$  to the problem described in definition 1. Notice that  $SP$  is a generalization of various shortest path problems. For instance, if we consider  $w^2 = 0$  and  $f = w^1 \cdot x$ , where  $\cdot$  is the dot product, this problem is the classical shortest path problem. If  $f = \min\{w^1 \cdot x : w^2 \cdot x \leq K\}$ , where  $K$  is a given number, this problem is the shortest weight-constrained path problem [7], and the decision version is known as ND30 in [7]. If  $f = \max\{w^1 \cdot x, w^2 \cdot x\}$ , the problem is the min-max shortest path problem [2,9,12] in literature.

We now give a formal definition of our problem, which is a combination of the two-machine flow shop problem and the shortest path problem.

**Definition 2.** Given any instance  $I$  of the shortest path problem  $SP$  with  $G = (V, A, w^1, w^2)$ , and each arc  $a_j \in A$  of  $I$  corresponds to a job  $J_j \in J$  with processing times  $p_{1j}$ ,  $p_{2j}$  on the two machines respectively. Define  $P_x$  be a set of jobs such that  $J_j \in P_x$  if and only if  $x_j = 1$ . The  $F2|\text{shortest path}|C_{max}$  problem is to find a feasible solution  $x$  of  $SP$ , that corresponds to a directed path connecting the vertices  $s$  and  $t$  in  $G$ , and assign the jobs of  $P_x$  on two-machine flow shop to minimize the makespan.

## 2.2 Johnson's Rule for Two-Machine Flow Shop Scheduling

In flow shop scheduling, a schedule is called a permutation schedule if all jobs are processed in the same order on each machine [4]. Johnson [8] proposed a sequencing rule for  $F2||C_{max}$ , which is referred as Johnson's rule in literature.

---

### Algorithm 1. Johnson's rule

---

- 1: Set  $S_1 = \{J_j \in J | p_{1j} \leq p_{2j}\}$  and  $S_2 = \{J_j \in J | p_{1j} > p_{2j}\}$ .
  - 2: Process the jobs in  $S_1$  first with a non-decreasing order of  $p_{1j}$ , and then schedule the jobs in  $S_2$  with a non-increasing order of  $p_{2j}$ , and ties may be broken arbitrarily.
- 

In Johnson's rule, jobs are scheduled as early as possible. This rule produces a permutation schedule, and Johnson showed that it is an optimal schedule. Notice that this schedule is delivered in  $O(n \log n)$  time.

We now introduce some well-known lower bounds for  $F2||C_{max}$ , that are used later to derive approximation algorithms to our problem. Let us denote by  $C_{max}$  the makespan in an arbitrary flow shop schedule with job set  $J$ , we have

$$C_{max} \geq \max \left\{ \sum_{J_j \in J} p_{1j}, \sum_{J_j \in J} p_{2j} \right\}, \quad (1)$$

and

$$C_{max} \leq \sum_{J_j \in J} (p_{1j} + p_{2j}). \quad (2)$$

For each job, we have

$$C_{max} \geq p_{1j} + p_{2j}, \quad \forall J_j \in J. \quad (3)$$

Suppose  $J_v$  is the critical job of the flow shop, we have

$$C_{max} = \max_{J_u \in J} \left\{ \sum_{j=1}^u p_{1j} + \sum_{j=u}^n p_{2j} \right\} = \sum_{j=1}^v p_{1j} + \sum_{j=v}^n p_{2j}. \quad (4)$$

### 2.3 Algorithms for Shortest Path Problems

In this paper, we use two following results of the shortest path problem.

The first one is the well-known Dijkstra's algorithm, which solves the classical shortest path problem with nonnegative edge weights in  $O(|V|^2)$  time [5].

The second one is an FPTAS result for min-max shortest path problem, which is presented by Aissi, Bazgan and Vanderpoorten [2]. Their algorithm, denoted as the ABV algorithm, is based on dynamic programming and scaling technique, and we have the following result.

**Theorem 1 ([2]).** *Given an arbitrary positive value  $\epsilon > 0$ , in a given directed graph with two nonnegative weights associated with each arc, a direct path  $P$  between two specific vertices can be found by the ABV algorithm with the property*

$$\max \left\{ \sum_{a_j \in P} w_j^1, \sum_{a_j \in P} w_j^2 \right\} \leq (1 + \epsilon) \max \left\{ \sum_{a_j \in P'} w_j^1, \sum_{a_j \in P'} w_j^2 \right\}$$

for any other path  $P'$ , and the running time is  $O(|A||V|^3/\epsilon^2)$ .

## 3 Computational Complexity of $F2|\text{shortest path}|C_{max}$

We argue that the decision version of our problem is NP-complete, by a reduction from a NP-complete problem PARTITION [7]. The proof is similar to the well-known NP-hardness proof of ND30 in [7], one could refer to the literature, such as the reduction presented in [3].

**Theorem 2.** *The decision problem of  $F2|\text{shortest path}|C_{max}$  is NP-complete.*

Nevertheless, we emphasize that ND30 is neither a special case nor simple application of our problem. Since idles may occur on machine 2 in the flow shop scheduling, it is not straightforward that the path found in our problem is relevant to a shortest weight-constrained path found in ND30, which has total weight and total length bounded by two given numbers.

## 4 Approximation Algorithms

### 4.1 A Natural Approximation Algorithm

The main idea of the first algorithm is, we first set  $w_j^1 = p_{1j} + p_{2j}$  and  $w_j^2 = 0$  for each arc and find the shortest path with respect to  $w^1$  by Dijkstra's algorithm. Then we schedule the corresponding jobs in the flow shop by Johnson's rule.

It is straightforward that the total running time of the JD algorithm is  $O(|V|^2)$ . Then we study the performance. First, we introduce some notations. Let  $J^*$  be set of jobs in an optimal solution, and  $C_{max}^*$  be the corresponding makespan.  $J_x$  and  $C_{max}$  are those returned by the JD algorithm.

**Algorithm 2.** The JD algorithm

- 
- 1: Find the shortest path in  $G$  with weight  $(w_j^1, w_j^2) := (p_{1j} + p_{2j}, 0)$  by Dijkstra's algorithm. For the returned solution  $x$ , construct the job set  $P_x$ .
  - 2: Schedule the jobs of  $P_x$  by Johnson's rule. Let  $\sigma$  be the returned job schedule and  $C_{max}$  the returned makespan, and denote the job set  $P_x$  by  $J_x$ .
  - 3: **return**  $J_x, \sigma$  and  $C_{max}$
- 

**Theorem 3.** *The JD algorithm is 2-approximate.*

*Proof.* By the lower bound (1) introduced in section 2.2, we have

$$2C_{max}^* \geq \sum_{J_j \in J^*} p_{1j} + \sum_{J_j \in J^*} p_{2j} = \sum_{J_j \in J^*} (p_{1j} + p_{2j}). \quad (5)$$

Since the returned path is shortest with respect to  $w^1$ , we have

$$C_{max} \leq \sum_{J_j \in J_x} (p_{1j} + p_{2j}) = \sum_{J_j \in J_x} w_j^1 \leq \sum_{J_j \in J^*} w_j^1 = \sum_{J_j \in J^*} (p_{1j} + p_{2j}), \quad (6)$$

Combining with (5) and (6), it follows that  $C_{max} \leq 2C_{max}^*$ .

Consider the following instance. A directed graph  $G$  includes three vertices, which are referred to  $v_1, v_2, v_3$ . There are three jobs (arcs):  $(v_1, v_2)$ ,  $(v_2, v_3)$ ,  $(v_1, v_3)$ , with processing times  $(1, 0), (0, 1), (2 - \epsilon, 0)$  respectively in which  $\epsilon$  is small enough. We wish to find a path from vertex  $v_1$  to  $v_3$ . The makespan of job schedule returned by the JD algorithm is  $C_{max} = 2 - \epsilon$  with the arc  $(v_1, v_3)$ , whereas the makespan of optimal job schedule is  $C_{max}^* = 1$  with the arcs  $(v_1, v_2), (v_2, v_3)$ . The bound is tight as  $\frac{C_{max}}{C_{max}^*} \rightarrow 2$  when  $\epsilon \rightarrow 0$ .  $\square$

## 4.2 An Improved Approximation Algorithm

Instead of finding a shortest path from  $s$  to  $t$  optimally with respect to certain weight, we could adopt the FPTAS result mentioned in section 2.3, that will return a  $(1 + \epsilon)$ -approximated solution for the min-max shortest path problem. Then we also implement Johnson's rule. In other words, by setting the objective function  $f = \max\{w^1 \cdot x, w^2 \cdot x\}$  in  $SP$ .

We initially set  $(w_j^1, w_j^2) := (p_{1j}, p_{2j})$ . The algorithm iteratively runs the above executions for the min-max shortest path problem and  $F2||C_{max}$  by adopting the following revision policy: in a current schedule, if there exists some job is big enough with respect to the current makespan, we will revise the weights of arcs corresponding to big jobs to  $(M, M)$ , where  $M$  is a sufficient large number, and then mark these jobs. The algorithm terminates if no such a job exists. Another terminating condition is that when a marked job appears in a current schedule. We return the schedule with minimum makespan among all current schedules as the solution of the algorithm. We denote this algorithm as the JAR algorithm.

**Algorithm 3.** The JAR algorithm

---

```

1: Initially,  $(w_j^1, w_j^2) := (p_{1j}, p_{2j})$ , for each arc  $a_j \in A$  corresponding to  $J_j \in J$ .
2: Given  $\epsilon > 0$ , implement the ABV algorithm to obtain a feasible solution  $x$  of  $SP$ , and construct the corresponding job set as  $P_x$ .
3: Schedule the jobs of  $P_x$  by Johnson's rule, denote the returned makespan as  $C'_{max}$ , and the job schedule as  $\sigma'$ .
4:  $J_x := P_x$ ,  $\sigma := \sigma'$ ,  $C_{max} := C'_{max}$ ,  $D := \emptyset$ ,  $M := (1 + \epsilon) \sum_{J_j \in J} (|p_{1j}| + |p_{2j}|) + 1$ .
5: while  $P_x \cap D = \emptyset$  and there exists a job  $J_j$  in  $P_x$  such that  $p_{1j} + p_{2j} > \frac{2}{3}C'_{max}$  do
6:   for all jobs with  $p_{1j} + p_{2j} > \frac{2}{3}C'_{max}$  in  $J \setminus D$  do
7:      $(w_j^1, w_j^2) := (M, M)$ ,  $D := D \cup \{J_j\}$ .
8:   end for
9:   Implement the ABV algorithm to obtain a feasible solution  $x$  of  $SP$ , and construct the corresponding job set as  $P_x$ .
10:  Schedule the jobs of  $P_x$  by Johnson's rule, denote the returned makespan as  $C'_{max}$ , and the job schedule as  $\sigma'$ .
11:  if  $C'_{max} < C_{max}$  then
12:     $J_x := P_x$ ,  $\sigma := \sigma'$ ,  $C_{max} := C'_{max}$ .
13:  end if
14: end while
15: return  $J_x$ ,  $\sigma$  and  $C_{max}$ .

```

---

Now, we discuss the computational complexity of the JAR algorithm. Let the total number of jobs be  $|A| = n$ . First, we need to revise the weights of at most  $n$  arcs, hence lines 6 - 8 execute at most  $O(n)$  times in the whole execution of our algorithm. And at least one job is added to  $D$  in each iteration, the iterations in lines 5 - 14 execute at most  $n$  times. In each iteration, the running time of obtaining a path by the ABV algorithm and a job schedule by Johnson's rule is  $O(n|V|^3/\epsilon^2)$  and  $O(n \log n)$  respectively. And  $O(n)$  time is enough to other operations. Hence, the total running time of the JAR algorithm is  $O(n^2(|V|^3/\epsilon^2 + \log n))$ .

The following theorem shows the performance of the JAR algorithm.

**Theorem 4.** *Given  $\epsilon > 0$ , the JAR algorithm is  $\frac{3}{2}(1 + \epsilon)$ -approximate.*

*Proof.* *Case 1.*  $J^* \cap D \neq \emptyset$

It implies that there is at least one job in the optimal solution, say  $J_j$ , such that  $(p_{1j} + p_{2j}) > \frac{2}{3}C'_{max}$  holds for a current schedule with makespan  $C'_{max}$  during the execution. Notice that the schedule returned by the JAR algorithm is the schedule with minimum makespan among all current schedules, and we have  $C_{max} \leq C'_{max}$ . It follows from (3) that

$$C_{max} \leq C'_{max} < \frac{3}{2}(p_{1j} + p_{2j}) \leq \frac{3}{2}C^*_{max}. \quad (7)$$

*Case 2.*  $J^* \cap D = \emptyset$

Consider the last current schedule during the execution of the algorithm. We denote the corresponding job set and the makespan as  $J'$  and  $C'_{max}$  respectively.

In this case, we first argue that  $J' \cap D = \emptyset$ . Suppose not, since  $J^* \cap D = \emptyset$ , the weights of arcs corresponding to the jobs in  $J^*$  have not been revised. Hence we have  $(1 + \epsilon) \max \left\{ \sum_{J_j \in J^*} w_j^1, \sum_{J_j \in J^*} w_j^2 \right\} < M$ . Moreover, by the assumption  $J' \cap D \neq \emptyset$ , we have  $\max \left\{ \sum_{J_j \in J'}, w_j^1, \sum_{J_j \in J'} w_j^2 \right\} \geq M$ . By Theorem 1, the solution returned by the ABV algorithm satisfies

$$M \leq \max \left\{ \sum_{J_j \in J'} w_j^1, \sum_{J_j \in J'} w_j^2 \right\} \leq (1 + \epsilon) \max \left\{ \sum_{J_j \in J^*} w_j^1, \sum_{J_j \in J^*} w_j^2 \right\} < M,$$

that leads to a contradiction.

Let  $J_v$  be a critical job in the last current schedule, and suppose that  $p_{1v} \geq p_{2v}$ . It follows from  $p_{1j} \geq p_{2j}$  for  $j = v + 1, \dots, n$  in the schedule returned by Johnson's rule and (4) that  $C'_{\max} \leq \sum_{J_j \in J'} p_{1j} + p_{2v}$ . Since  $J' \cap D = \emptyset$ , we have  $p_{1j} + p_{2j} \leq \frac{2}{3}C'_{\max}$  for all jobs  $J_j \in J'$ , as otherwise the algorithm will continue. Thus, it follows from (1), (3), Theorem 1 and the fact that the schedule returned by the JAR algorithm is the schedule with minimum makespan among all current schedules, we have

$$\begin{aligned} C_{\max} &\leq C'_{\max} \leq \sum_{J_j \in J'} p_{1j} + p_{2v} \leq \sum_{J_j \in J'} w_j^1 + \frac{1}{2}(p_{1v} + p_{2v}) \\ &\leq (1 + \epsilon) \max \left\{ \sum_{J_j \in J^*} w_j^1, \sum_{J_j \in J^*} w_j^2 \right\} + \frac{1}{3}C'_{\max} \\ &= (1 + \epsilon) \max \left\{ \sum_{J_j \in J^*} p_{1j}, \sum_{J_j \in J^*} p_{2j} \right\} + \frac{1}{3}C'_{\max} \\ &\leq (1 + \epsilon)C^*_{\max} + \frac{1}{3}C'_{\max}. \end{aligned}$$

It suffices to show that  $C_{\max} \leq C'_{\max} \leq \frac{3}{2}(1 + \epsilon)C^*_{\max}$ .

For the case that the last current schedule with critical job  $p_{1v} < p_{2v}$ , an analogous argument will yield the same result. Therefore, the JAR algorithm is  $\frac{3}{2}(1 + \epsilon)$ -approximate for  $F2|\text{shortest path}|C_{\max}$ .

The following instance shows that the worst case ratio of the JAR algorithm can not less than  $\frac{3}{2}$ . A directed graph  $G$  has four vertices, which are referred to  $v_1, v_2, v_3, v_4$ . Given  $\epsilon > 0$ , there are four jobs (arcs):  $(v_1, v_2)$ ,  $(v_1, v_3)$ ,  $(v_3, v_2)$ , and  $(v_2, v_4)$ , with processing times  $(1, 1)$ ,  $((1 + 4\epsilon), 0)$ ,  $(0, (1 + 4\epsilon))$  and  $(1, 1)$  respectively. We wish to find a path from  $v_1$  to  $v_4$ . Notice that the ABV algorithm returns the path with the arcs  $(v_1, v_2)$  and  $(v_2, v_4)$ , and the corresponding makespan  $C'_{\max}$  by Johnson's rule is 3. All the corresponding jobs satisfy  $p_{1j} + p_{2j} = 2 \leq \frac{2}{3}C'_{\max}$ , and thus the algorithm terminates. Therefore, the makespan of the returned job schedule by the JAR algorithm is  $C_{\max} = 3$ . On the other hand, the optimal makespan is  $C^*_{\max} = 2 + 4\epsilon$ , with the corresponding

arcs  $(v_1, v_3)$ ,  $(v_3, v_2)$ , and  $(v_2, v_4)$ . The worst case ratio of the JAR algorithm can not less than  $\frac{3}{2}$  as  $\frac{C_{max}}{C_{max}^*} \rightarrow 3/2$  when  $\epsilon \rightarrow 0$  for this instance.  $\square$

## 5 Conclusions

This paper studies a combination problem of two-machine flow shop scheduling and shortest path problems. It is interesting to find an approximation algorithm with a better performance ratio for this problem. On the other hand, one can consider the combination problem of more generalized forms, such as combining with  $m$ -machine flow shop scheduling problem, or covering problem presented in [11]. All these questions motivate us to further investigate.

**Acknowledgments.** This work has been supported by Bilateral Scientific Co-operation Project between Tsinghua University and K.U. Leuven. We would like to thank Fabrice Talla Nobibon for helpful comments and suggestions.

## References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall, New Jersey (1993)
2. Aissi, H., Bazgan, C., Vanderpoorten, D.: Approximating min-max (Regret) versions of some polynomial problems. In: Chen, D.Z., Lee, D.T. (eds.) COCOON 2006. LNCS, vol. 4112, pp. 428–438. Springer, Heidelberg (2006)
3. Batagelj, V., Brandenburg, F.J., Mendez, P., Sen, A.: The generalized shortest path problem. CiteSeer Archives (2000)
4. Chen, B., Potts, C.N., Woeginger, G.J.: A review of machine scheduling: Complexity, algorithms and approximability. In: Du, D.Z., Pardalos, P.M. (eds.) Handbook of Combinatorial Optimization, vol. 3, pp. 21–169. Kluwer (1998)
5. Dijkstra, E.W.: A note on two problems in connexion with graph. Numerische Mathematik 1, 269–271 (1959)
6. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research 1, 117–129 (1976)
7. Gary, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, San Francisco (1979)
8. Johnson, S.M.: Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly 1, 61–68 (1954)
9. Kouvelis, P., Yu, G.: Robust discrete optimization and its applications. Kluwer Academic Publishers, Boston (1997)
10. Wang, Z., Cui, Z.: Combination of parallel machine scheduling and vertex cover. Theoretical Computer Science 460, 10–15 (2012)
11. Wang, Z., Hong, W., He, D.: Combination of parallel machine scheduling and covering problem. Working paper. Tsinghua University (2012)
12. Yu, G.: Min-max optimization of several classical discrete optimization problems. Journal of Optimization Theory and Applications 98, 221–242 (1998)

# The Program Download Problem: Complexity and Algorithms

Chao Peng<sup>1,\*</sup>, Jie Zhou<sup>1</sup>, Binhai Zhu<sup>2</sup>, and Hong Zhu<sup>1</sup>

<sup>1</sup> Software Engineering Institute, East China Normal University,  
3663 Zhongshan North Rd., Shanghai, 200062, China

{cpeng@sei.ecnu.edu.cn, jiezhou@shnu.edu.cn, hzhu@fudan.edu.cn}

<sup>2</sup> Department of Computer Science, Montana State University,  
Bozeman, MT 59717-3880, USA  
{bhz@cs.montana.edu}

**Abstract.** In this paper, we consider the Program Download Problem (PDP) which is to download a set of desired programs from multiple channels. When the problem is to decide whether the download can be done by a given deadline  $d$  and each program appears in each of the  $n$  channels at most once, denoted as  $PDP(n, 1, d)$ , we prove that  $PDP(n, 1, d)$  is NP-Complete by a reduction from 3-SAT(3). We can extend the NP-hardness proof to  $PDP(2, 2, d)$  where there are only two channels but each program could appear in each channel at most twice. We show that the aligned version of the problem (APDP) is polynomially solvable by reducing it to a maximum flow problem. For a different version of the problem, MPDP, where the objective is to maximize the number of program downloaded before a given deadline  $d$ , we prove that it is fixed-parameter tractable.

**Keywords:** Program Download Problem, NP-Complete, FPT Algorithm, Approximation Algorithm.

## 1 Introduction

The last decades has witnessed several information technology trends, such as cloud computing, Internet applications, big data, as well as the integration of telecommunications networks, cable TV networks and the Internet. However, the client-server approach is still prevalent for information service. In this paper we will study how a client can quickly download his/her required contents from a server which broadcasts contents through multiple wireless or wired channels. We

---

\* This research is partially supported by the Innovation Program of Shanghai Municipal Education Commission, the Natural Science Foundation of China under Grant No.91118008 and Grant No.61232006, the national high-tech research and development plan of China under grant No.2011AA010101, the Shanghai Knowledge Service Platform Project (No.ZF1213) and ECNU Project "Heterogenous Network Convergence Technologies for CPS". We are grateful to Dr. Jian Li for helpful discussions on the Aligned Program Download Problem.

encounter this problem very often in many important applications such as Video-on-Demand (VoD), distance learning, digital libraries, electronic commerce, etc.

Take VoD for example, usually it is implemented by a client-server architecture supported by certain transport networks such as CATV or Internet[1,2]. In a VoD system, the server simply broadcasts segments of a popular video periodically on several channels. A large number of clients can be served simultaneously by downloading their requested programs from these channels [3]. By this fashion, the bandwidth savings can be considerable since a few (10 or 20) very popular videos are likely to account for nearly 80% of demands [4].

Lots of researchers have designed algorithms to compute the broadcast schedule in the server side, so that a user will be guaranteed to get the next segment while playing the current one, thus the whole video can be played out continuously [5,6]. But little consideration has been paid on the client side, for they usually assume that a user can download from all channels simultaneously.

However, this assumption is not true for most cases in a wireless communication system [7]. For example, future vehicular networking aims to build an Internet of Vehicles by enabling a vehicle to be connected with other vehicles or the Road Side Unit (RSU) such as 802.11b/p access point [8]. In such cyber-physical systems, an RSU can act as not only an access point but also a data server between vehicles. It can keep broadcasting data such as value-added advertisement, local news, real-time traffic and digital map etc, vehicles passing by will tune in these channels to seek what they are interested in.

When a vehicle is downloading programs from multiple channels, it needs to switch between different channels which contain its requested contents. Such a decision is not easy to make for there might be conflicts and tradeoffs, thus we need efficient algorithms to schedule the download process. There are some existing works on similar topics, during the time when we were conducting research on this paper, Lu et al. [9,10] independently designed data retrieval scheduling methods for multi-Channel wireless data broadcast environments.

In this paper, we formulate the data retrieving issue into a Program Download Problem, which studies from a more general perspective. The rest of this paper is organized as follows: Section 2 presents the Program Download Problem (PDP), proves the NP-Completeness of PDP and a special case where there are only two channels but a specific program could appear at most twice in each channel. Section 3 describes the Aligned Program Download Problem (APDP) and proposes a polynomial time algorithm to solve it. In section 4, we study the Maximum Program Download Problem (MPDP) and we prove that it is fixed-parameter tractable. Finally, we conclude the paper in Section 5.

## 2 Complexity of the Program Download Problem

We first define the general problem we will investigate in this paper.

**Problem:** *Program Download Problem (PDP).*

**Instance:** In a broadcasting scheme, there are  $|C| = n$  channels broadcasting contents in a finite program set  $U$ , each program in  $U$  has a unit time interval length, two consecutive programs could be separated by some arbitrary time interval (or junk program).  $S \subseteq U$  is a target set of programs to be downloaded and  $d$  is a positive number representing a deadline. For any channel  $c_i \in C$ , there are at most  $m$  programs, possibly separated by junk programs. Each program broadcasted in a channel has a starting time and a finishing time, a user can download any program from its beginning and can switch between different channels at the end of a program without additional time, but he/she can only download from one channel at one time.

**Question:** Given such an instance  $(C, U, S, n, m, d)$ , is there a schedule that can collect all programs in  $S$  before time  $d$ ?

To make our presentation easier, we consider an extra parameter, i.e., the number of times  $p$  a program can repeat in a channel. Together with the number of channels  $n$  and the deadline  $d$ , we can denote the PDP problem as  $PDP(n, p, d)$ .

**Theorem 1.** *Even when each program appears in each of the  $n$  channels at most once, the Program Download Problem is NP-Complete.*

**Proof.** The theorem really states that  $PDP(n, 1, d)$  is NP-complete.  $PDP(n, 1, d)$  is obviously in NP, since given an instance  $PDP(n, 1, d)$  and a corresponding download schedule as the certificate, we need only to check whether i) all programs in  $S$  are in the schedule, ii) no program in the schedule overlap with another one, and iii) all programs in the schedule finish before  $d$ . All three steps can be done in polynomial time with respect to  $n$  and  $m$ .

Next we focus on reducing a known NP-Complete problem to  $PDP(n, 1, d)$ . We choose 3-SAT(3), which is a special 3-SAT instance where each clause contains at most three literals and each variable appears in the input clauses 3 times, twice positively and once negatively (or vice versa) [11,12]. Let the input formula  $\phi$  contain  $N$  variables  $x_i, 1 \leq i \leq N$  and  $M$  clauses  $C_1, C_2, \dots, C_M$ .

For each variable  $x_i$ , we construct two channels  $c_i, \bar{c}_i$  corresponding to  $x_i, \bar{x}_i$  respectively. WLOG, assume that  $x_i$  appears in  $\phi$  twice (say in  $C_{i1}, C_{i2}$ ) and  $\bar{x}_i$  appears in  $\phi$  once (say in  $C_{i3}$ ). The clause programs to be downloaded correspond to the clauses containing  $x_i$  and  $\bar{x}_i$ . Specifically, for  $x_i$  we construct two programs  $C_{i1}, C_{i2}$  each with length one and starts at  $4(i-1)+0.5$  and  $4(i-1)+1.5$ , followed with a unit-length peg program  $P_i$ , starting at time  $4(i-1)+3$ . Starting at time  $4n$  we put a list of unit-length programs, containing all the clause programs except  $C_{i1}, C_{i2}$  and all peg programs except  $P_i$ .

For  $\bar{x}_i$ , we construct a unit-length program  $C_{i3}$  in channel  $\bar{c}_i$  which starts at time  $4(i-1)+1$ , followed with a unit-length program  $P_i$ , starting at time  $4(i-1)+3$ . At both channels  $c_i$  and  $\bar{c}_i$ , in the time intervals  $[0, 4(i-1)+0.5]$ ,  $[4(i-1)+2.5, 4(i-1)+3]$ , and in the time interval  $[4i, 4n]$  we can insert some arbitrary junk programs which are not in  $U$ . The programs to be downloaded include all clause programs and peg programs. It is easily seen that each program can appear in each channel exactly once.

Note that when  $x_i$  appears once and  $\bar{x}_i$  appears twice, then the above construction is symmetric. Moreover, one cannot download  $\{C_{i1}, C_{i2}\}$  and  $\{C_{i3}\}$  at the same time. Finally, as for the peg program  $P_i$ , it appears in every other channel, except  $c_i$  and  $\bar{c}_i$ , after time  $4n$ . Therefore, to download all the required  $P_i$ 's before time  $4n$ , one must download it through channel  $c_i$  or  $\bar{c}_i$ .

We next show that  $\phi$  is satisfiable iff all the required programs (clause and peg programs) can be downloaded by time  $d = 4n$ .

( $\rightarrow$ ) If  $\phi$  is satisfiable with some truth assignment, then for  $x_i = \text{True}$  which appears in  $C_{i1}$  and  $C_{i2}$  we download  $C_{i1}$  and  $C_{i2}$  in channel  $c_i$ ; for  $x_i = \text{False}$  which appears in  $C_{i3}$  we download  $C_{i3}$  from channel  $\bar{c}_i$ . (If  $x_i$  appears once in  $\phi$  and  $\bar{x}_i$  appears twice in  $\phi$ , the details are similar.)  $P_i$  can be downloaded from either  $c_i$  or  $\bar{c}_i$ . It is clear that by time  $4n$  we have got all required programs.

( $\leftarrow$ ) If by time  $4n$  all clause and peg programs have been downloaded, the first argument is that we cannot download clause programs in channel  $c_i$  and  $\bar{c}_i$ , before time  $4n$ , at the same time. This naturally gives us the truth assignment: if we download some clause program in  $c_i$  (the corresponding clause contains  $x_i$ ), then  $x_i$  will be assigned  $\text{True}$ ; if we download some clause program in  $\bar{c}_i$  (the corresponding clause contains  $\bar{x}_i$ ), then  $x_i$  will be assigned  $\text{False}$ . (If no clause program in  $c_i$  and  $\bar{c}_i$  is downloaded, then assign  $x_i$  either  $\text{True}$  or  $\text{False}$ .)

The above reduction takes  $O(mn)$  time. Hence the theorem is proven.  $\square$

If there is no restriction on the length of a continuous sequence of "junk" programs (i.e., those programs not in  $U$ ), then we can put any continuous sequence of such programs (say, with a total length of  $X$ ) starting at time  $4n$  in all the channels. Then if  $\phi$  is satisfiable, we can download all required programs by time  $4n$ ; if  $\phi$  is unsatisfiable, we can download all the required program by time  $4n + X + 1$  the earliest. The approximation factor is at least  $(4n + X + 1)/4n$ , which could be arbitrarily large if  $X$  is unbounded. As for the time interval between two consecutive programs, we can also limit it to be less than one unit without change the NP-Completeness of  $PDP(n, 1, d)$ .

**Corollary 1.** *The optimization version of  $PDP(n, 1, d)$  does not admit any factor  $1 + \frac{X}{4n}$  approximation unless  $P = NP$ , where  $X$  is the length of the maximum continuous sequence of junk programs.*

If there are two channels and each program to be downloaded appears at most once in each channel, we can use dynamic programming to solve the problem  $PDP(2, 1, d)$  in polynomial time. However, the above theorem implies that we can prove the NP-completeness of  $PDP(2, 2, d)$ , i.e., when the number of channels is limited to be 2 and the number of times a program can appear in a channel is at most 2.

**Corollary 2.**  *$PDP(2, 2, d)$  is NP-Complete.*

**Proof.** It is easy to check that  $PDP(2, 2, d)$  is in NP. So we will focus on reducing 3-SAT(3) to  $PDP(2, 2, d)$ .

The process of reduction is almost the same as the above theorem, except that we will have only two channels now, one channel  $c^+$  for the positive occurrences

of variables and another channel  $c^-$  for the negative occurrences of variables. We will construct a corresponding unit-length program for each clause in a 3-SAT(3) instance.

For each variable  $x_i$ , we can assume that  $x_i$  appears in  $\phi$  twice (say in  $C_{i1}, C_{i2}$ ) and  $\bar{x}_i$  appears in  $\phi$  once (say in  $C_{i3}$ ). Now we put the two unit-length programs corresponding to  $C_{i1}, C_{i2}$  on channel  $c^+$  at  $4(i-1) + 0.5$  and  $4(i-1) + 1.5$  respectively, followed with a unit-length peg program  $P_i$  begin at time  $4(i-1)+3$ . On channel  $c^-$ , we construct a unit-length program corresponding to  $C_{i3}$  which starts at time  $4(i-1) + 1$ , followed with a unit-length program  $P_i$ , starting at time  $4(i-1) + 3$ . On the other hand, if  $x_i$  has one positive occurrence and two negative occurrences, we will similarly put one corresponding program on  $c^+$  and the other two programs on  $c^-$ .

On both channels, starting at time  $4n$  we put a list of unit-length programs corresponding to all clauses. Similarly, we can show that  $\phi$  is satisfiable iff all the required programs can be downloaded by time  $4n$  (the deadline  $d$  is still equal to  $4n$ ).

Notice that each (clause) program appears in  $c^+$  (resp.  $c^-$ ) at most three times, as a clause could contain three positive (resp. negative) literals. However, we could easily show that Not-All-Equal 3-SAT(3), i.e., each clause must contain at least one positive literal and one negative literal, is still NP-complete. In that case, each program could appear in  $c^+$  (or  $c^-$ ) at most twice. Finally, the above reduction takes  $O(MN)$  time. Hence the corollary is proven.  $\square$

### 3 The Aligned Program Download Problem

In PDP, the junk programs (or idle time intervals) could be of any length. If we restrict that the length of the junk programs must be a unit as well, then if two programs have a conflict then their corresponding time intervals must be identical. We hence have the following variation of the problem.

**Problem:** *Aligned Program Download Problem (APDP)*.

**Instance:** In a broadcasting scheme, there are  $|C| = n$  channels each broadcasting  $m$  programs in a finite set  $U$ , each program (including those junk programs not in  $U$ ) has a unit time interval length and can appear in a channel multiple times.  $S \subseteq U$  is a target set of programs to be downloaded.

**Question:** Given such an instance  $(C, U, S, n, m, d)$ , can one collect all programs in  $S$  from channels in  $C$  at the end of all channels, i.e., by time  $d$ ?

We show that APDP can be solved in polynomial time by reducing it to a maximum flow problem [13].

**Theorem 2.** *APDP is in P.*

**Proof.** We can solve APDP by transforming it into a maximum flow problem. Given an instance  $(C, U, S, n, m, d)$  of APDP, we first remove all programs later than  $d$ , then create a node  $s_i (1 \leq i \leq |S|)$  for each program in  $S$ , a node  $r_t (1 \leq t \leq m)$  for each time unit  $t$  and a node  $c_{jt} (1 \leq j \leq n, 1 \leq t \leq m)$  for

each program broadcasted in channel  $c_j$  at time unit  $t$ . Then we draw a directed edge from node  $s_i$  to  $c_{jt}$  iff their corresponding programs are the same. We also draw a directed edge from node  $c_{jt}$  to  $r_t$  for each program broadcasted at time unit  $t$ . Next we create a source node  $a$  and connect it to all the  $s_i$  nodes. Finally we create sink node  $b$  and connect all nodes  $r_t$  ( $1 \leq t \leq m$ ) to  $b$ .

We assign a unit capacity to each edge, next we show that  $(C, U, S, n, m, d)$  is a yes instance of APDP iff we can find an  $|S|$  flow from  $a$  to  $b$  in the graph.

( $\rightarrow$ ) If we can download all programs in  $S$  before  $d$ , suppose some  $s_i$  is located on channel  $j$  at time  $t$  in the solution schedule, then we can send a unit flow from  $a$  to node  $s_i$ , node  $c_{jt}$ , node  $r_t$  and finally to node  $b$ . All such flows will not meet each other at a  $s_i$  node or a  $c_{jt}$  node, and neither will they meet at a  $r_t$  node, since no two programs will be downloaded at the same time. Thus the paths for each flow will be disjoint with others, which means there is no conflict between these flows and they add up to  $|S|$  units in total [14].

( $\leftarrow$ ) If there is a  $|S|$  flow from  $a$  to  $b$  in the constructed graph, it is not difficult to find that this flow can be decomposed into  $|S|$  disjoint unit flows since each edge in this graph has a unit capacity. Now we pick out all programs corresponding to some  $c_{jt}$  node in each unit flow, combine them together and we will get  $S$  that can be collected before  $d$ .

The whole process takes polynomial time in  $n + m$  and we can easily turn the above description into an algorithm. Hence the theorem is proven.  $\square$

## 4 The Maximum Program Download Problem

As we know, under the assumption that  $P \neq NP$ , it is unlikely to find efficient, exact, and deterministic algorithms for NP-complete problems [15]. So if the complexity is measured in terms of the input size only, an exact algorithm for an NP-complete problem requires super-polynomial running time. However, some NP-complete problems admit efficient FPT (fixed-parameter tractable) algorithms, i.e., with some parameter  $k$  the algorithm runs in  $f(k)n^{O(1)}$  steps, where  $f(-)$  is any function on the parameter  $k$  [16].

In this section we design an FPT algorithm for PDP. To distinguish from the previous versions of the problem, when the instance is general but the objective is to maximize the number of programs downloaded, we call the corresponding problem *Maximum Program Download Problem*, abbreviated as MPDP. (It is easy to see that Theorem 1 holds for MPDP as well since we really have  $S = U$  in the proof. So MPDP is also NP-complete.) In the parameterized version of MPDP, the parameter to be considered is the size of the target set  $S$ , which will be denoted as  $k$ . We adopt a dynamic programming approach to design the algorithm. A table  $Z$  with  $2^k$  records will be used to memorize the partial solutions. Given a subset  $X \subseteq S$ , we can denote it by a binary sequence of length  $k$  in which a bit  $i$  is set to 1 iff the program  $s_i$  is in  $X$ . Thus all subsets of  $S$  can be denoted by a sequence of  $2^k$  numbers, from 0 to  $2^k - 1$ . In table  $Z$ ,  $Z[X]$  will record the earliest time ( $Z[X][0]$ ) to download all programs in the subset denoted by  $X$  and the corresponding channel ( $Z[X][1]$ ) for the last

program to be downloaded in this subset. Starting from  $Z[0]$ , we can fill the table in the following way: when  $Z[X]$  is computed, we check every program which is broadcasted after  $Z[X][0]$  and is not in the subset  $X$  yet, then we choose the first appearance of each program  $i$  and update  $Z[X+2^i]$ . When  $Z[2^k-1]$  is computed, we can trace back the whole schedule and return it as the solution. Since every step is strictly computed to ensure there is no conflict for downloading, we are sure that each computed  $Z[X]$  records a corresponding schedule to download the subset denoted by  $X$ .

The above statement directly establishes the correctness of the following algorithm for finding the target set  $S$  with a limited size  $k$  in MPDP.

**Algorithm 1.** **MPDP-FPTalgo**( $C, U, S, n, m, d$ )

```

1   if there is a program  $s_i \in S$  that is not included in any channel before  $d$ ,
2       then return "No such schedule!" ;
3   Initialize  $Z[0] \leftarrow \{0, 0\}$ ,  $\alpha \leftarrow 0$ ,  $k \leftarrow |S|$ , remove all programs after  $d$ ;
4   Initialize  $L$  as an empty list and  $Q$  to be a queue with one element  $Z[0]$ ;
5   for all  $i$  from 1 to  $2^k - 1$  do  $Z[0] \leftarrow \{\infty, 0\}$  ;
6   while  $Q$  is not empty
7       Take a node  $Z[i]$  from  $Q$ ;
8       for all  $s_j$  that is later than  $Z[i][0]$  and the  $j$ -th bit of  $i$ 's binary code is 0
9           Find the first  $s_j$ , let it be on channel  $u$  and finish at time  $t$ ;
10          if  $t < Z[i + 2^j][0]$  then update  $Z[i + 2^j] \leftarrow \{t, u\}$  ;
11          if  $Q$  is empty
12              then  $\alpha \leftarrow \alpha + 1$ , add to  $Q$  all updated  $Z[i^+]$  with  $\alpha$  "1" bits in  $i^+$ ;
13          if  $Z[2^k - 1] = \{\infty, 0\}$  then return "No such schedule!" ;
14           $i \leftarrow 2^k - 1$ ; /* trace back the download schedule */
15        while  $i != 0$ 
16            Take  $s_y$  located on channel  $Z[i][1]$  from time  $Z[i][0] - 1$  to  $Z[i][1]$ ;
17            Add  $\{s_y, Z[i][1], Z[i][0] - 1\}$  to  $L$ ;
18             $i \leftarrow i - 2^y$ ;
19    return ( $L$ ) ;

```

**Theorem 3.** *MPDP is fixed-parameter tractable.*

**Proof.** The above algorithm MPDP-FPTalgo first finds partial solutions for subsets of  $S$  with size 1 first, then based on these partial solutions it deals with subsets of  $S$  with size 2 and 3 ... until  $k$ , which is the final solution to be computed. When a new element is added to  $Q$ , let it be  $Z[i]$  with  $\alpha$  "1" bits in  $i$ 's binary code, then 1) all subsets of  $S$  with fewer than  $\alpha$  elements have been checked and, 2) it is added together with all other  $Z[j]$  with  $\alpha$  "1" bits in  $j$ . By 1), we make sure that  $Z[i]$  has been correctly computed. By 2), we make sure all possibilities will be taken into consideration. Thus when there is a schedule for MPDP, the algorithm will find it out and update  $Z[2^k - 1]$ . And when the algorithm has updated  $Z[2^k - 1]$ , it can trace the corresponding solution schedule for PDP and return it back.

Now let us check the complexity of this algorithm. The size of table  $Z$  and queue  $Q$  are both no more than  $2^k$ , while the space used for all channels will be  $O(mn)$ . The cycle from line 6 to line 12 contributes a main part of time complexity, which might check through at most  $2^k$  items in  $Q$  and each item might take  $O(mn)$  time to compute its successors (subsets with one more program). On the other hand, we might need  $O(mn \log(mn))$  time to sort all programs in the channels. Thus the whole time complexity of this algorithm is  $O(2^k mn \log(mn))$ , which is of the form  $f(k) \cdot (mn)^{O(1)}$  and we can conclude that MPDP is fixed-parameter tractable.  $\square$

## 5 Conclusion

In this paper we prove that the Program Download Problem, aiming at minimizing the finishing time, is NP-Complete by reducing 3-SAT(3) to it. We further prove the NP-Completeness of a special case where there are only two channels but a specific program could appear at most twice in each channel. We find that the Aligned Program Download Problem (APDP) is in P and we proposed a polynomial time algorithm to solve it. For Maximum Program Download Problem (MPDP) aiming at maximizing the number of programs downloaded, we prove that it is fixed-parameter tractable by designing an FPT algorithm for it.

## References

1. Almeroth, K.C., Ammar, M.H.: The use of multicast delivery to provide a scalable and interactive Video-on-Demand service. *IEEE Journal on Selected Areas in Communications* 14(5), 1110–1122 (1996)
2. Viswanathan, S., Imielinski, T.: Metropolitan area Video-on-Demand service using pyramid broadcasting. *Multimedia Systems* 4(4), 197–208 (1996)
3. Peng, C., Tan, Y., Xiong, N.X., Yang, L.T., Park, J.H., Kim, S.S.: Adaptive Video-on-Demand Broadcasting in Ubiquitous Computing Environment. *Journal of Personal and Ubiquitous Computing* 13(7), 479–488 (2009)
4. Aggarwal, C.C., Wolf, J.L., Yu, P.S.: A permutation-based pyramid broadcasting scheme for Video-on-Demand systems. In: Proc. International Conference on Multimedia Computing and Systems, pp. 118–126 (June 1996)
5. Hua, K.A., Sheu, S.: Skyscraper broadcasting: a new broadcasting scheme for metropolitan Video-on-Demand systems. In: Proc. ACM SIGCOMM 1997 Conference, Cannes, France, pp. 89–100 (September 1997)
6. Juhn, L., Tseng, L.: Harmonic broadcasting for Video-on-Demand service. *IEEE Trans. on Broadcasting* 43(3), 268–271 (1997)
7. Inoue, M., Ohnishi, M., Peng, C., Li, R., Morino, H.: NerveNet: A Future Regional Platform Network for Various Context-Aware Services with Sensors and Actuators. *IEICE Trans. on Communications* E94-B(3), 618–629 (2011)
8. Karagiannis, G., Altintas, O., Ekici, E., Heijenk, G., Jarupan, B., Lin, K., Weil, T.: Vehicular Networking: A Survey and Tutorial on Requirements, Architectures, Challenges, Standards and Solutions. *IEEE Communications Surveys & Tutorials* 13(4), 584–616 (2011)

9. Lu, Z.X., Shi, Y., Wu, W.L., Fu, B.: Efficient Data Retrieval Scheduling for Multi-Channel Wireless Data Broadcast. In: Proceedings of the 31st IEEE International Conference on Computer Communications (INFOCOM), pp. 891–899 (2012)
10. Lu, Z.X., Wu, W.L., Fu, B.: Optimal Data Retrieval Scheduling in the Multi-Channel Data Broadcast Environments. *IEEE Trans. on Computers* (2012)
11. Tovey, C.A.: A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics* 8(1), 85–89 (1984)
12. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley, New York (1994)
13. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Networks Flows. Prentice-Hall, NJ (1993)
14. Peng, C., Shen, H.: A New Approximation Algorithm For Computing 2-Restricted Disjoint Paths. *IEICE Transactions on Information and Systems* E90-D(2), 465–472 (2007)
15. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, New York (1979)
16. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer (1999)

# Finding Theorems in NBG Set Theory by Automated Forward Deduction Based on Strong Relevant Logic

Hongbiao Gao, Kai Shi, Yuichi Goto, and Jingde Cheng

Department of Information and Computer Sciences,

Saitama University, Saitama 338-8570, Japan

{gao,shikai,gotoh,cheng}@aise.ics.saitama-u.ac.jp}

**Abstract.** Automated theorem finding is one of 33 basic research problems in automated reasoning which was originally proposed by Wos in 1988, and it is still an open problem. For the problem, Cheng has proposed a forward deduction approach based on strong relevant logic. To verify the effectiveness of the approach, we tried to rediscover already known theorems in NBG set theory by using the approach, and succeeded in rediscovery of several known theorems. However, the method of the rediscovery is ad hoc, but not systematic. This paper gives an analysis and discussion for our experiment method and results from the viewpoint of the systematic method. The paper also presents some issues and future research directions for a systematic method of automated theorem finding based on Cheng's approach.

**Keywords:** Automated theorem finding, forward deduction, strong relevant logic, NBG set theory.

## 1 Introduction

The problem of automated theorem finding (ATF for short) is one of 33 basic research problems in automated reasoning which was originally proposed by Wos in 1988 [10,11], and it is still an open problem until now.

The most important and difficult requirement of the problem is that, in contrast to proving conjectured theorems supplied by the user, it asks for criteria that an automated reasoning program can use to find some theorems in a field that must be evaluated by theorists of the field as new and interesting theorems. The significance of solving the problem is obvious because an automated reasoning program satisfying the requirement can provide great assistance for scientists in various fields [3].

ATF cannot be solved by the current automated theorem proving approach, and the reasoning is the only way to fit for ATF [3]. Reasoning is the process of drawing new conclusions from some premises which are known facts and/or assumed hypotheses. In contrast, proving is the process of finding a justification for an explicitly specified statement from given premises, which are already

known facts or previously assumed hypotheses. Discovery is the process to find out or bring to light of that which was previously unknown. For any discovery, the discovered thing and its truth must be unknown before the completion of discovery process. Because reasoning is the only way to draw new, previously unknown conclusions from given premises, there is no discovery process that does not invoke reasoning [4].

However, not all logics can serve well as the fundamental logic underlying ATF. As a well-known fact, the classical mathematical logic (CML for short) and its various extensions are not suitable for ATF because they have the well-known “implicational paradoxes. [4]” In order to avoid the “implicational paradoxes”, relevant logics T, E, and R were constructed [1,2]. However, there are still some logical theorems in the relevant logics which are not natural in the sense of entailment. Cheng named them “conjunction-implicational paradoxes” and “disjunction-implicational paradoxes [4].”

Cheng proposed strong relevant logics, named Tc, Ec, and Rc, for conditional relation representation and reasoning. As a modification of T, E, and R, Tc, Ec and Rc reject all conjunction-implicational and disjunction-implicational paradoxes in T, E and R, respectively, and therefore, by using strong relevant logics as the fundamental logic to underlie ATF, one can avoid those problems in using CML, various extensions of CML, and relevant logics T, E and R. Cheng also proposed predicate strong relevant logics, named TcQ, EcQ, and RcQ [4].

To solve the ATF problem, a forward deduction approach based on the strong relevant logics was proposed by Cheng [3]. To verify the effectiveness of the approach, we presented a case study of ATF in von Neumann-Bernays-Godel (NBG) set theory by automated forward deduction based on the strong relevant logics [7]. NBG set theory was chosen for the case study, because it is the foundation of mathematics and other nature science fields, and the approach we used can be also hopeful for other mathematics and natural science fields. In the case study, by using Cheng’s approach, we rediscovered some known theorems of NBG set theory. The ultimate goal of ATF is to find new and interesting theorems. From the viewpoint of the mechanism and process of deducing theorems, “rediscovery” is as same as “discovery”. Therefore, if we have found known theorems by using Cheng’s approach, it is hopeful that we can also use the approach to find new and interesting theorems.

However, we used some ad hoc methods in our experiment. First, we chose some known theorems as target theorems. Besides, we used several necessary axioms of strong relevant logic and NBG set theory as premises to deduce them, but not use all of axioms. Second, we prepared some necessary theorems to deduce target known theorems by using substitution but not forward deduction. Therefore, our experiment method was not a systematic method based on Cheng’s approach for ATF.

This paper gives an analysis of our experiment results and discussion for our experiment method from a viewpoint of systematic method. The paper also presents some issues and future research directions of systematic method based on Cheng’s approach. The rest of the paper is organized as follows: Section 2

explains the logic-based reasoning and ATF terminology used in the case study. Section 3 presents how we made this case study, and shows our experiment results. Section 4 gives an analysis and discussion about the case study. Section 5 gives some ideas of the future research directions from the systematic method view for ATF. Finally, some concluding remarks are given in Section 6.

## 2 Automated Forward Deduction Based on Strong Relevant Logic

A formal logic system  $L$  is an ordered pair  $(F(L), \vdash_L)$  where  $F(L)$  is the set of well formed formulas of  $L$ , and  $\vdash_L$  is the consequence relation of  $L$  such that for a set  $P$  of formulas and a formula  $C$ ,  $P \vdash_L C$  means that within the framework of  $L$  taking  $P$  as premises we can obtain  $C$  as a valid conclusion.  $Th(L)$  is the set of logical theorems of  $L$  such that  $\varphi \vdash_L T$  holds for any  $T \in Th(L)$ . According to the representation of the consequence relation of a logic, the logic can be represented as a Hilbert style system, a Gentzen sequent calculus system, or a Gentzen natural deduction system [4].

Let  $(F(L), \vdash_L)$  be a formal logic system and  $P$  be a non-empty set of sentences. A formal theory with premises  $P$  based on  $L$ , called a  $L$ -theory with premises  $P$  and denoted by  $T_L(P)$ , is defined as  $T_L(P) =_{df} Th(L) \cup Th_L^e(P)$  where  $Th_L^e(P) =_{df} \{A | P \vdash_L A \text{ and } A \notin Th(L)\}$ ,  $Th(L)$  and  $Th_L^e(P)$  are called the logical part and the empirical part of the formal theory, respectively, and any element of  $Th_L^e(P)$  is called an empirical theorem of the formal theory [4].

Based on the definition above, the problem of ATF can be said as “for any given premises  $P$ , how to construct a meaningful formal theory  $T_L(P)$  and then find new and interesting theorem in  $Th_L^e(P)$  automatically?” [3]

The notion of degree [5] is defined as follows: Let  $\theta$  be an arbitrary  $n$ -ary ( $1 \leq n$ ) connective of logic  $L$  and  $A$  be a formula of  $L$ , the degree of  $\theta$  in  $A$ , denoted by  $D_\theta(A)$ , is defined as follows: (1)  $D_\theta(A) = 0$  if and only if there is no occurrence of  $\theta$  in  $A$ , (2) if  $A$  is in the form  $\theta(a_1, a_2, \dots, a_n)$  where  $a_1, a_2, \dots, a_n$  are formulas, then  $D_\theta(A) = \max\{D_\theta(a_1), D_\theta(a_2), \dots, D_\theta(a_n)\} + 1$ , (3) if  $A$  is in the form  $\sigma(a_1, a_2, \dots, a_n)$  where  $\sigma$  is a connective different from  $\theta$  and  $a_1, a_2, \dots, a_n$  are formulas, then  $D_\theta(A) = \max\{D_\theta(a_1), D_\theta(a_2), \dots, D_\theta(a_n)\}$ , and (4) if  $A$  is in the form  $QB$  where  $B$  is a formula and  $Q$  is the quantifier prefix of  $B$ , then  $D_\theta(A) = D_\theta(B)$ .

The degree of logic fragment [5] is defined as follows: Let  $\theta_1, \theta_2, \dots, \theta_n$  be connectives of logic  $L$  and  $k_1, k_2, \dots, k_n$  be natural numbers, the fragment of  $L$  about  $\theta_1, \theta_2, \dots, \theta_n$  and their degrees  $k_1, k_2, \dots, k_n$ , denoted by  $Th^{(\theta_1, k_1, \theta_2, k_2, \dots, \theta_n, k_n)}(L)$ , is a set of logical theorems of  $L$  which is inductively defined as follows (in the terms of Hilbert style axiomatic system): (1) if  $A$  is an axiom of  $L$  and  $D_{\theta_1}(A) \leq k_1, D_{\theta_2}(A) \leq k_2, \dots, D_{\theta_n}(A) \leq k_n$ , then  $A \in Th^{(\theta_1, k_1, \theta_2, k_2, \dots, \theta_n, k_n)}(L)$ , (2) if  $A$  is the result of applying an inference rule of  $L$  to some members of  $Th^{(\theta_1, k_1, \theta_2, k_2, \dots, \theta_n, k_n)}(L)$  and  $D_{\theta_1}(A) \leq k_1, D_{\theta_2}(A) \leq k_2, \dots, D_{\theta_n}(A) \leq k_n$ , then  $A \in Th^{(\theta_1, k_1, \theta_2, k_2, \dots, \theta_n, k_n)}(L)$ , (3) Nothing else are in  $Th^{(\theta_1, k_1, \theta_2, k_2, \dots, \theta_n, k_n)}(L)$ . Similarly, the notion of degree of formal theory about conditional can also be

generally extended to other logic connectives, and the degree of the fragment of a formal theory with premises P based on the fragment  $Th^{(\theta_1, k_1, \theta_2, k_2, \dots, \theta_n, k_n)}(L)$  of a logic system L is also similarly defined as the notion of degree of the logic fragment.

The deduction distance of a theorem in a forward deduction is the length of the longest deductive path from a premise node to the theorem node in the deduction tree of the theorem. The deduction tree of a certain theorem is recursively constructed as follows: (1) See the certain theorem as the root node of the tree. (2) See the premises which deduce the root node by using inference rules as the parent nodes of the root node. (3) See each premise node as a new root node.

Automated forward deduction is a process of deducing new and unknown conclusions automatically by applying inference rules to premises and previously deduced conclusions repeatedly until some previously specified condition is satisfied.

FreeEnCal [5] is a forward reasoning engine for general purpose, and is hopeful candidate to do forward deduction based on strong relevant logic automatically, because it can provide an easy way to customize reasoning task by providing different axioms, inference rules and facts. Users can set the degree of logical operators to make FreeEnCal to reason out in principle all logical theorem schemata of the fragment  $Th^{(\theta_1, k_1, \theta_2, k_2, \dots, \theta_n, k_n)}(L)$ . Based on the logical theorem schemata of the fragment  $Th^{(\theta_1, k_1, \theta_2, k_2, \dots, \theta_n, k_n)}(L)$ , FreeEnCal can also reason out in principle all empirical theorems satisfying the conditions about degrees from given empirical premises.

### 3 The Case Study of Automated Theorem Finding in NBG Set Theory

The purpose of the experiment [7] is to verify whether or not the forward deduction approach based on strong relevant logics is hopeful for ATF. The experiment process is divided into four steps. At the first step, we investigated which logical theorems and axioms of NBG set theory were necessary to deduce target known theorems chosen from Quaife's book [8]. At the second step, by using FreeEnCal, we deduced several logic fragments which contained those necessary logical theorems. At the third step, we chose several necessary axioms of NBG set theory as empirical premises, and based on those logic fragments to deduce empirical theorems. Finally, we investigated whether or not the target theorems could be found in the empirical theorems deduced by FreeEnCal.

As a result, we rediscovered eight known theorems of NBG set theory which were proved by Quaife (in Quaife's book, the following "V" means the set) [8]: (1) theorem PO1:  $\forall x(x \subseteq x)$ . (2) theorem EQ1:  $\forall x(x = x)$ . (3) theorem I2:  $\forall x\forall y(x \cap y = y \cap x)$ . (4) theorem I6:  $\forall x(x \cap x = x)$ . (5) theorem CP2:  $\forall x\forall y\forall u\forall v((\langle u, v \rangle \in (x \times y)) \Rightarrow (\langle v, u \rangle \in (y \times x)))$ . (6) theorem UP1:  $\forall x\forall y(\{x, y\} = \{y, x\})$ . (7) theorem SS1:  $\forall x(\{x\} \in V)$ . (8) theorem OP1 :

$\forall x \forall y (< x, y > \in V)$ . Our experiment results show that Cheng's approach is hopeful for ATF.

## 4 Discussion for the Case Study

There are three issues in the case study from the viewpoint of systematic method: (1) How can we make the logic fragments we used concise to deduce interesting empirical theorems as many as possible in the acceptable time and memory space? (2) How can we choose those empirical theorems whose terms should be substituted, and substitute their terms automatically? (3) How can we present some criteria by which computer programs can find interesting theorems from deduced empirical theorems automatically?

The first issue is how to make the logic fragments concise. It is not practical to use all the logic theorems of whole logic fragments to deduce empirical theorems, because it will spend too much time and huge memory space. To find interesting empirical theorems as many as possible in the acceptable time and memory space, we have to present a method to find those logical theorems which are not useful for deducing empirical theorems or interesting empirical theorems and remove them from the deduced logical fragments. In our case study, we only inputted several necessary axioms of strong relevant logic to deduce several little logic fragments and based on them rediscovered those target theorems, which implies us to find one interesting theorems, it is not necessary to use all the logic theorems in one logic fragment  $Th^{(\theta_1, k_1, \theta_2, k_2, \dots, \theta_n, k_n)}(EcQ)$ , and we can find interesting empirical theorems based on the concise logic fragments.

The second issue is how to choose those empirical theorems whose terms should be substituted, and how to substitute their terms automatically. FreeEn-Cal as a general forward deduction engine can not make substitutions of terms by functions automatically, and if we only use forward deduction approach for ATF, lots of interesting theorems cannot be found. We have to use the substitution method combined with forward deduction approach. However, it is not practical to substitute each term of each empirical theorem with all of functions and/or individual constants, because it will spend huge time and memory space. We have to present a method to choose those empirical theorems whose terms should be substituted. In our case study, we substituted some terms with functions, for example, to rediscover theorem EQ1, we did not input axioms of NBG set theory:  $\forall x \forall y ((x \subseteq y) \wedge (y \subseteq x)) \Rightarrow (x = y)$ , but inputted the instance of the axiom:  $\forall x ((x \subseteq x) \wedge (x \subseteq x)) \Rightarrow (x = x)$ . However, it is sure that the ad hoc method help us succeed in rediscovery of the known theorems, which implies the substitutions of terms are necessary for ATF, and we have to solve the limitation problem of FreeEnCal and/or the forward deduction approach.

The third issue is how to present some criteria for “new and interesting theorems”. For the discovery of unknown theorems, we have to present some criteria for “interesting theorems” so that we can find them automatically by computer from lots of deduced empirical theorems. In our case study, as rediscovery of known theorems, it is not necessary for us to present some criteria such that

automated reasoning program can find interesting theorems automatically, because it is not difficult to search them from deduced empirical theorems sets.

## 5 Discussion for the Research Directions of Systematic Method

Based on the discussion for the issues of our case study, we presented some primitive ideas for the future research directions of the systematic methods.

For the first issue, we have found a primitive idea to make the logic fragment concise. By analyzing deduced logic fragments in our case study, we found that not all of the logical theorems can be used when we deduce empirical theorems. Logical theorems of the logic fragments we used are classified into three kinds of style theorems:  $L_1 \Rightarrow L_2$  style logical theorems,  $L_1 \wedge L_2$  style logical theorems, and  $L_1$  style logical theorems. For example,  $\forall x \forall y \forall z (A \Rightarrow B) \Rightarrow \forall x \forall y \forall z ((C \Rightarrow A) \Rightarrow (C \Rightarrow B))$  is a  $L_1 \Rightarrow L_2$  style logical theorem.  $\forall x \forall y (\forall z (A \Rightarrow B) \Rightarrow \forall z ((C \Rightarrow A) \Rightarrow (C \Rightarrow B)))$  is a  $L_1$  style logical theorem. When a formula is deduced by modus ponens from a logic theorem A and other formula, only  $L_1 \Rightarrow L_2$  style logic theorem is used for the logic theorem A. In other words, the logic theorem A must not be  $L_1 \wedge L_2$  style logical theorems, and  $L_1$  style logical theorems. When we choose strong relevant logic as our logic basic tool to do forward deduction, besides the generalization of axioms, modus ponens is the only one inference rule [4]. Therefore, we can remove those  $L_1 \wedge L_2$  style and/or  $L_1$  style logical theorems from our logic fragments so that we can deduce empirical theorems based on concise logical fragments.

As for the second issue, we can make an environment which can provide an interactive method between computers and scientists so that scientists can make use of their specific knowledge in specific field to do substitution with functions and/or individual constants in time, because even if scientific discovery processes can be simulated by computer programs automatically, the specific knowledge of scientists is the power of the programs [4]. Besides, the environment should search those empirical theorems which can deduce new empirical theorems by substitution and indicate to scientists, For example, if the following empirical theorems  $\forall x \forall y ((x = y) \Rightarrow (x \subseteq y))$  and  $\forall x ((x = (x \cap x))$  exist in empirical theorems deduced by FreeEnCal, the environment should automatically search them and indicate to scientists that if term “y” is substituted by function “ $x \cap x$ ”, a new empirical theorem “ $x \subseteq (x \cap x)$ ” will be found.

As for the third issue, we present filtering methods by which we can remove uninteresting theorems from deduced empirical theorems step by step and provide the rest theorems as candidates of interesting theorems for scientists.

First, lots of empirical theorems with quantifiers can be classified into same group. For example: (1)  $\forall x \forall y \forall u \forall v ((< u, v > \in (x \times y)) \Rightarrow (< v, u > \in (y \times x)))$ . (2)  $\forall x \forall y \forall u (\forall v (< u, v > \in (x \times y)) \Rightarrow \forall v (< v, u > \in (y \times x)))$ . (3)  $\forall x \forall y (\forall u \forall v (< u, v > \in (x \times y)) \Rightarrow \forall u \forall v (< v, u > \in (y \times x)))$ . (4)  $\forall x (\forall y \forall u \forall v (< u, v > \in (x \times y)) \Rightarrow \forall y \forall u \forall v (< v, u > \in (y \times x)))$ . (5)  $\forall x \forall y \forall u \forall v (< u, v > \in (x \times y)) \Rightarrow \forall x \forall y \forall u \forall v (< v, u > \in (y \times x))$ . The theorem  $((< u, v > \in (x \times y)) \Rightarrow (< v, u > \in (y \times x)))$  is a

representation of the above theorems. Therefore, to conveniently analyze empirical theorems, we removed the quantifiers of all the empirical theorems and called those theorems “core empirical theorems” of deduced empirical theorems. It is enough to analyze “core empirical theorems”. If the core empirical theorem is an interesting theorem, then we can see the primitive theorems as candidates of interesting theorems. The filtering method is hopeful for removing those uninteresting empirical theorems, because we have analyzed about 400 known theorems in NBG set theory and we found the most of interesting theorems are “interesting” in the predicates part but not in their quantifiers domain.

The second filtering method is to check whether a formula includes a tautological sub-formula or not. We translated each core empirical theorem into patterns according to different predicates and different terms, for example, the theorem  $((x = x) \Rightarrow (x = x)) \Rightarrow ((x \subseteq x))$  can be translated by  $(A \Rightarrow A) \Rightarrow B$ . We conjecture if one theorem contains a tautology part, this empirical theorem must not be an interesting empirical theorem. For example,  $((x = x) \Rightarrow (x = x)) \Rightarrow ((x \subseteq x))$  cannot be called an interesting theorem, because it is like  $(A \Rightarrow A) \Rightarrow B$  which contains a tautology  $A \Rightarrow A$ .

Besides the filtering methods proposed by us, those theorems needing longer deduction distance are more likely to be interesting theorems, because interesting theorems are always holding complex functions and predicates, which need longer deduction distance to be deduced.

By using the filtering methods to analyze empirical theorems deduced in our case study, besides those known theorems, we have also found the following interesting theorems. (1)  $\forall x \forall y ((y \in x) \Rightarrow (y \in (x \cap x)))$ . (2)  $\forall x \forall y \forall z (\neg(z \in \{x, y\}) \Rightarrow (\neg(z = x) \wedge \neg(z = y)))$ . (3)  $\forall x (x \subseteq (x \cap x))$ . (4)  $\forall x \forall y ((x \cap y) \subseteq (y \cap x))$ . (5)  $\forall x \forall y \forall z (z \in (x \cap y) \Rightarrow z \in (y \cap x))$ . (6)  $\forall x \forall y (x = y \Rightarrow y = x)$ .

Epistemic programming [4] and its environment EPLAS [6,9] are hopeful to combine proposed solutions for the issues and forward deduction process with FreeEnCal. On one hand, by using EPLAS, scientists can make programming to make logic fragments concise and find new and interesting theorems from the empirical theorem sets as an ordered epistemic processes, i.e., to find new and interesting theorems step by step. On the other hand, EPLAS can provide an interactive method between computers and scientists, by which scientists can make use of specific knowledge in specific fields for ATF, like the substitution of terms.

## 6 Concluding Remarks

We have given an analysis and discussion about our case study in which we have rediscovered several theorems of NBG set theory by Cheng’s approach. We have presented the filtering methods to remove uninteresting theorems step by step, and by our filtering methods, besides those rediscovered known theorems, some other interesting theorems were also found by Cheng’s approach. We also showed some future research directions for the systematic method of ATF.

There are many interesting and challenging research problems in our future works. First, in this case study, to make the experiment can be finished in short

time, we only used several axioms of NBG set theory, we should use more axioms of NBG set theory as premises to deduced theorems in future works. Second, we can also try to find some interesting or known theorems by using Cheng's approach in other theory or other fields like Peano's Arithmetic, graph theory, and combinatorics. Finally, we will combine Cheng's epistemic programming approach with automated forward deduction approach to find new and interesting theorems in the next step work.

## References

1. Anderson, A.R., Belnap Jr., N.D.: *Entailment: The Logic of Relevance and Necessity*, vol. 1. Princeton University Press (1975)
2. Anderson, A.R., Belnap Jr., N.D., Dunn, J.M.: *Entailment: The Logic of Relevance and Necessity*, vol. 2. Princeton University Press (1992)
3. Cheng, J.: Entailment Calculus as the Logical Basis of Automated Theorem Finding in Scientific Discovery. In: *Systematic Methods of Scientific Discovery: Papers from the 1995 Spring Symposium*, pp. 105–110. AAAI Press - American Association for Artificial Intelligence (1995)
4. Cheng, J.: A Strong Relevant Logic Model of Epistemic Processes in Scientific Discovery. In: *Information Modelling and Knowledge Bases XI, Frontiers in Artificial Intelligence and Applications*, vol. 61, pp. 136–159 (2000)
5. Cheng, J., Nara, S., Goto, Y.: FreeEnCal: A Forward Reasoning Engine with General-Purpose. In: Apolloni, B., Howlett, R.J., Jain, L. (eds.) *KES 2007, Part II. LNCS (LNAI)*, vol. 4693, pp. 444–452. Springer, Heidelberg (2007)
6. Fang, W., Takahashi, I., Goto, Y., Cheng, J.: Practical Implementation of EPLAS: An Epistemic Programming Language for All Scientists. In: Proc. 10th International Conference on Machine Learning and Cybernetics. The IEEE Systems, Man, and Cybernetics Society, pp. 608–616 (2011)
7. Gao, H., Shi, K., Goto, Y., Cheng, J.: Automated Theorem Finding by Forward Deduction Based on Strong Relevant Logic: A Case Study in NBG Set Theory. In: Proc. 11th International Conference on Machine Learning and Cybernetics. The IEEE Systems, Man, and Cybernetics Society, pp. 1859–1865 (2012)
8. Quaife, A.: *Automated Development of Fundamental Mathematical Theories*. Kluwer Academic (1992)
9. Takahashi, I., Nara, S., Goto, Y., Cheng, J.: EPLAS: An Epistemic Programming Language for All Scientists. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) *ICCS 2007, Part I. LNCS*, vol. 4487, pp. 406–413. Springer, Heidelberg (2007)
10. Wos, L.: *Automated Reasoning: 33 Basic Research Problem*. Prentice-Hall (1988)
11. Wos, L.: The Problem of Automated Theorem Finding. *Journal of Automated Reasoning* 10(1), 137–138 (1993)

# Perturbation Analysis of Maximum-Weighted Bipartite Matchings with Low Rank Data

Xingwu Liu<sup>1,\*</sup> and Shang-Hua Teng<sup>2,\*\*</sup>

<sup>1</sup> Institute of Computing Technology, Chinese Academy of Sciences

<sup>2</sup> Department of Computer Science, Viterbi School of Engineering, USC

**Abstract.** In this paper, we partially address a question raised by David Karger [5] regarding the structure of maximum-weighted bipartite matchings when the affinity data is of low rank. The affinity data of a weighted bipartite graph  $G$  over the vertex sets  $U = V = \{1, \dots, n\}$  means the  $n \times n$  matrix  $W$  whose entry  $W_{ij}$  is the weight of the edge  $(i, j)$  of  $G$ ,  $1 \leq i, j \leq n$ .  $W$  has rank at most  $r$  if there are  $2r$  vector  $\mathbf{u}_1, \dots, \mathbf{u}_r, \mathbf{v}_1, \dots, \mathbf{v}_r \in \mathbb{R}^n$  such that

$$W = \sum_{i=1}^r \mathbf{u}_i \mathbf{v}_i^T.$$

In particular, we study the following locality property of the matchings: For an integer  $k > 0$ , we say the *locality* of  $G$  is at most  $k$  if for every matching  $\pi$  of  $G$ , either  $\pi$  has the maximum weight, or its weight is smaller than that of another matching  $\sigma$  with  $|\pi \setminus \sigma| \leq k$  and  $|\sigma \setminus \pi| \leq k$ .

We prove the following theorem: For every  $W \in [0, 1]^{n \times n}$  of rank  $r$  and  $\epsilon \in [0, 1]$ , there exists  $\tilde{W} \in [0, 1]^{n \times n}$  such that (i)  $\tilde{W}$  has rank at most  $r + 1$ , (ii)  $\max_{i,j} |W_{i,j} - \tilde{W}_{i,j}| \leq \epsilon$ , and (iii) the weighted bipartite graph with affinity data  $\tilde{W}$  has locality at most  $\lceil r/\epsilon \rceil^r$ .

## 1 Introduction

The maximum-weighted bipartite matching problem is a fundamental problem at the intersection of graph theory, geometric design, and combinatorial optimization [7,4]. Its input instance is a bipartite graph  $G = (U, V, W)$  where  $U = V = \{1, 2, \dots, n\}$  and each entry  $W_{ij}$  of the matrix  $W$  defines the weight of the edge  $(i, j)$ , and its objective is to find a perfect matching of the maximum weight. This is equivalent to the following problem: given a matrix  $W \in \mathbb{R}^{n \times n}$ , find a permutation  $\pi \in S_n$  – where  $S_n$  stands for the symmetric group of  $n$  elements – such that the *value* of  $\pi$  relative to  $W$ , which is defined to be

$$v_W(\pi) = \sum_{i=1}^n W_{i,\pi(i)},$$

is maximized.

\* The work was done when the author was visiting USC. The work was partially supported by National Natural Science Foundation of China (61173009).

\*\* Supported in part by the NSF grants 1111270 and 0964481.

It is well known that when  $\pi$  is not maximum relative to  $W$ , there exists a sequence of indices  $S = i_0 i_1 \dots i_{t-1}$  such that the permutation

$$(S \circ \pi)(i) = \begin{cases} \pi(i) & \text{if } i \notin S \\ \pi(i_{(j+1) \bmod t}) & \text{if } i = i_j \end{cases},$$

improves  $\pi$ . We sometimes refer to the sequence  $S$  as an *augmenting sequence* of  $\pi$  relative to  $W$ . Note that if  $S$  is an augmenting sequence of  $\pi$  relative to  $W$ , so is the sequence  $i_j \dots i_{t-1} i_0 i_1 \dots i_{j-1}$  for any  $1 \leq j \leq t-1$ .

In 2009, in a personal conversation with the second author, David Karger [5] asked to characterize the structure of matchings when the affinity data of a weighted bipartite graph is of low rank. The present paper tries the first step to study this problem. For an integer  $r$ ,  $W$  has rank at most  $r$  if there exist  $2r$  vectors  $\mathbf{u}_1, \dots, \mathbf{u}_r, \mathbf{v}_1, \dots, \mathbf{v}_r \in \mathbb{R}^n$  such that

$$W = \sum_{i=1}^r \mathbf{u}_i \mathbf{v}_i^T.$$

To warm up, consider the case when  $r = 1$ , i.e., there exist real vectors  $\mathbf{u} = (u_1, \dots, u_n)^T$  and  $\mathbf{v} = (v_1, \dots, v_n)^T$  such that  $W = \mathbf{u}\mathbf{v}^T$ . We say that a permutation  $\sigma$  sorts the entries of  $\mathbf{u}$  in descending order if

$$u_{\sigma(1)} \geq u_{\sigma(2)} \geq \dots \geq u_{\sigma(n)}.$$

Then, it is relatively standard to show that  $\pi$  is a maximum permutation if and only if the following is true: Suppose  $\sigma$  is the permutation that sorts the entries of  $\mathbf{u}$  in descending order, then  $\pi \circ \sigma$  also sorts the entries of  $\mathbf{v}$  in descending order. Thus, the augmenting structure of the permutations relative to a rank-one affinity matrix has the following property: For any non-optimum permutation  $\pi$ , there exists a sequence  $S = i_0 i_1$  which improves  $\pi$ .

**Definition 1 (Locality of a matrix).** *The locality  $L(W)$  of a matrix  $W \in \mathbb{R}^{n \times n}$  is defined to be the minimum  $k$  such that every non-optimum permutation relative to  $W$  has an augmenting sequence of length at most  $k$ .*

The above example means that a rank-1 matrix has locality at most two, independent of its size! We make the following conjecture, which arises naturally from the rank-1 example. The study of this conjecture has eventually led us to the main theorem of this paper.

*Conjecture 1 (Low Rank Bipartite Matchings).* There exists a function  $h : \mathbb{Z}_+ \rightarrow \mathbb{Z}_+$  such that for all  $n$ , every matrix  $W \in \mathbb{R}^{n \times n}$  has locality at most  $h(\text{rank}(W))$ .

In this paper, we show that although Conjecture 1 is false, it is nearly true from the perspective of perturbations. More specifically, we prove the following theorem regarding the locality of low-rank matrices.

**Theorem 1 (Main).** *For every  $W \in [0, 1]^{n \times n}$  and  $\epsilon \in [0, 1]$ , there exists  $\tilde{W} \in [0, 1]^{n \times n}$  such that  $\text{rank}(\tilde{W}) \leq \text{rank}(W) + 1$ ,*

$$\max_{i,j} |W_{i,j} - \tilde{W}_{i,j}| \leq \epsilon,$$

*and the locality of  $\tilde{W}$  is at most  $\lceil \frac{\text{rank}(W)}{\epsilon} \rceil^{\text{rank}(W)}$ .*

As a byproduct, it is shown that the conjecture is true for matrices defined over a finite set  $D$ , in particular, the locality of every  $W \in D^{n \times n}$  is at most  $|D|^{\text{rank}(W)}$ .

We hope that our work may lead to an alternative approach to fast approximation of maximum weighted bipartite matchings [3] by approximating the affinity data with low-rank data. A low-rank based approach could also be useful for geometrically defined matching problems [9,10,8,1,6]. For example, in our next step of research, we hope to have a better understanding of graph decomposition – i.e., approximating a weighted graph as the sum of a ‘low-rank’ graph and a graph whose structure is easy for matching augmentation.

The rest of this paper is organized as follows: In Section 2, we present a counter-example to Conjecture 1. We show that for any integer  $n \geq 2$ , there is a matrix  $W \in [0, 1]^{n \times n}$  of rank 2 which has a permutation that can be augmented only by a sequence of length  $n$ . In Section 3, we prove Theorem 1 by showing that every  $r$ -rank matrix, up to an  $\epsilon$ -perturbation, has locality at most  $\lceil r/\epsilon \rceil^r$ . Section 4 concludes this paper.

## 2 A Counter-Example to Conjecture 1

In this section, we show a counter-example to the Conjecture 1. The objective is to construct, for each positive integer  $n$ , a rank-2 matrix  $W \in [0, 1]^{n \times n}$  such that there is a non-optimum permutation all of whose augmenting sequences relative to  $W$  have length  $n$ .

Our counter-example matrix takes the form  $W = \mathbf{u} \cdot \mathbf{u}^T + H_\lambda$ , where  $\mathbf{u} = (1, (n-1)/n, \dots, 1/n)^T \in [0, 1]^{n \times 1}$  and  $H_\lambda \in [0, 1]^{n \times n}$  has all entries equal to zero except that the entry at the first row and the  $n^{th}$  column is equal to a parameter  $\lambda$ . The value  $\lambda$  will be determined later in this section.

Specifically,

$$W_{ij} = \begin{cases} 1/n + \lambda & \text{if } i = 1, j = n \\ (n-i+1)(n-j+1)/n^2 & \text{otherwise} \end{cases} \quad (1)$$

Note that  $\text{rank}(W) = 2$  if  $\lambda \neq 0$ , and otherwise  $\text{rank}(W) = 1$ . We now show that if  $\lambda$  is properly chosen, then there is a non-optimal permutation all of whose augmenting sequences relative to  $W$  have length  $n$ .

We first review a structural property of the permutations for rank-one matrices.

**Proposition 1.** For any positive integer  $k$ , assume  $A = \mathbf{u} \cdot \mathbf{v}^T$ , where  $\mathbf{u}, \mathbf{v} \in D^{k \times 1}$  with  $D \subseteq \mathbb{R}$ . Assume further that  $u_i \geq u_{i+1} + \delta, v_i \geq v_{i+1} + \delta$ , for  $1 \leq i \leq k-1, \delta > 0$ . Then, (i) the identity permutation  $I$  is the unique optimum permutation of  $A$ , and (ii) any non-optimum permutation  $\pi$  satisfies  $v_A(\pi) \leq v_A(I) - \delta^2$ .

*Proof.* As entries of  $\mathbf{u}$  and  $\mathbf{v}$  are distinct, the optimum permutation of  $A$  is unique and is equal the identity permutation. To prove (ii), note that for arbitrary real numbers  $x, y, u, v$  satisfying  $x - y \geq \delta, u - v \geq \delta$ , we have  $xu + yv \geq xv + yu + \delta^2$ , from which part (ii) of the proposition follows.  $\square$

We now establish a basic property of matrix  $W$ .

**Lemma 1.** Let  $W$  be the matrix defined by Equation (1) and  $I_n$  be the identity permutation in  $S_n$ . Then, for any  $\pi \in S_n$  such that  $\pi \neq I_n$  and  $\pi(1) \neq n$ , we have  $v_W(\pi) \leq v_W(I_n) - \frac{1}{n^2}$ .

*Proof.* Recall that  $W = \mathbf{u} \cdot \mathbf{u}^T + H_\lambda$ , where  $\mathbf{u} = (1, (n-1)/n, \dots, 1/n)^T \in [0, 1]^{n \times 1}$ . Let  $B = \mathbf{u} \cdot \mathbf{u}^T$ . Then, the optimum permutation of  $B$  is  $\pi_B = I_n$ . Since  $\pi(1) \neq n$ , we have  $v_W(\pi) = v_B(\pi)$ . By Proposition 1,  $v_B(\pi) \leq v_B(\pi_B) - \frac{1}{n^2}$ . Thus,  $v_W(\pi) = v_B(\pi) \leq v_B(\pi_B) - \frac{1}{n^2} = v_B(I_n) - \frac{1}{n^2} = v_W(I_n) - \frac{1}{n^2}$ .  $\square$

We now know that Conjecture 1 is not true.

**Theorem 2.** Let  $W$  be the matrix defined by Equation (1). There exists  $\lambda \in [0, 1 - 1/n]$  such that  $W$  has a non-optimal permutation that has no augmenting sequence of length  $l$ , for any  $l < n$ .

*Proof.* For any  $\pi \in S_n$  with  $\pi(1) = n$ , we have  $v_W(\pi) = v_B(\pi) + \lambda$ , where  $B = \mathbf{u} \cdot \mathbf{u}^T$ . Let

$$\pi_0 = \operatorname{argmax}_{\pi \in S_n, \pi(1)=n} \{v_B(\pi)\}.$$

One can verify that

$$\pi_0(i) = \begin{cases} n & \text{if } i = 1 \\ i - 1 & \text{otherwise} \end{cases} \quad (2)$$

We now choose  $\lambda$  such that  $v_W(\pi_0) = v_W(I_n) - 1/(2n^2)$ . By Proposition 1, we know  $\lambda \in [0, 1 - 1/n]$ , implying that  $W \in [0, 1]^{n \times n}$ .

For any  $\pi \in S_n$  with  $\pi \neq I_n$ , if  $\pi(1) = n$ , then  $v_W(\pi_0) \geq v_W(\pi)$  by the definition of  $\pi_0$ . If  $\pi(1) \neq n$ , then by Lemma 1, we have  $v_W(\pi) \leq v_W(I_n) - 1/n^2 < v_W(I_n) - 1/(2n^2) = v_W(\pi_0)$ ; the last equality holds due to the definition of  $\lambda$ . Thus,  $I_n$  is the unique optimum permutation of  $W$ , and  $v_W(\pi_0) \geq v_W(\pi)$  for any non-optimum  $\pi \in S_n$ .

Therefore,  $\pi_0$  is the desired permutation, since it is the second-best permutation of  $W$  and can be improved only by an augmenting sequence of length  $n$ . An example of its augmenting sequence is  $S = 12 \cdots n$ .  $\square$

### 3 The Perturbation Theorem

While Theorem 2 means that Conjecture 1 is false, we show in this section that every low-rank matrix can be approximated by another low-rank matrix whose locality only depends on the rank of the original matrix and the magnitude of the perturbation.

We start with a few lemmas that will be useful for the analysis in this section.

**Lemma 2.** *For any set  $D$  of real numbers where  $d = |D| < \infty$  and any positive integer  $n$ , any matrix  $W \in D^{n \times n}$  of rank  $r$  has at most  $d^r$  distinct rows.*

*Proof.* Since  $\text{rank}(W) = r$ , there is a submatrix  $M \in D^{r \times r}$  that has full rank. Without loss of generality, assume that  $M$  is located at the upper-left corner of  $W$ , that is,  $M$  is the principle sub-matrix  $W[1 : r, 1 : r]$ . Let  $\alpha_i = (W_{i1}, W_{i2}, \dots, W_{in}) \in D^{1 \times n}$ , the  $i^{\text{th}}$  row of  $W$ , and  $\beta_i = (W_{i1}, W_{i2}, \dots, W_{ir})$ , for  $1 \leq i \leq n$ .

By the property of  $M$ , we know that for each  $1 \leq j \leq n$ , there is a unique real vector  $\lambda_j \in \mathbb{R}^{1 \times r}$  such that

$$\alpha_j = \lambda_j \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \dots \\ \alpha_r \end{pmatrix},$$

implying that  $\beta_j = \lambda_j M$ , so  $\lambda_j = \beta_j M^{-1}$ . Hence, for any  $1 \leq i, j \leq n$ , if  $\beta_i = \beta_j$ , we have  $\lambda_i = \lambda_j$ , which means that  $\alpha_i = \alpha_j$ . Consequently, the number of distinct rows of  $W$  is exactly that of distinct  $\beta$ 's. The lemma follows immediately from the fact that there are at most  $d^r$  distinct  $\beta$ 's.  $\square$

**Lemma 3.** *If  $W \in \mathbb{R}^{n \times n}$  has at most  $s$  distinct rows, its locality is at most  $s$ .*

*Proof.* For an arbitrary non-optimum matching  $\pi \in S_n$  of  $W$ , let  $P = i_0 i_1 \dots i_{k-1}$  be one of the shortest augmenting sequence of  $\pi$ . We will prove that  $k \leq s$ .

Without loss of generality, we assume that  $\pi^{-1}(i_j) = j$  for all  $0 \leq j \leq k-1$ . Then,

$$v_W(\pi) < v_W(P \circ \pi) = v_W(\pi) + \sum_{m=0}^{k-1} (W_{m, i_{m+1} \bmod k} - W_{m, i_m}),$$

implying that  $\sum_{m=0}^{k-1} (W_{m, i_{m+1} \bmod k} - W_{m, i_m}) > 0$ .

Borrowing the notation used in the proof of Lemma 2, we now show that the  $k$  rows  $\alpha_0, \dots, \alpha_{k-1}$  are pairwise different.

For contradiction, assume that  $\alpha_j = \alpha_l$  for some  $0 \leq j < l \leq k-1$ . Again without loss of generality, we further assume that  $0 = j < l < k-1$ : otherwise an alternative augmenting sequence of length  $k$  can be used so that  $j$  and  $l$  meet this requirement.

Now we show that the augmenting sequence  $P$  can be divided into two subsequences, so that at least one subsequence also augments  $\pi$ .

Specifically, define two sequences  $P_1 = i_0 i_{l+1} i_{l+2} \dots i_{k-1}$  and  $P_2 = i_1 i_2 \dots i_l$ .

$$\begin{aligned} v_W(P_1 \circ \pi) + v_W(P_2 \circ \pi) &= 2v_W(\pi) + \sum_{m \in \{0, \dots, k-1\} \setminus \{0, l\}} (W_{m, i_{m+1} \bmod k} - W_{m, i_m}) \\ &\quad + W_{0, i_{l+1}} - W_{0, i_0} + W_{l, i_1} - W_{l, i_l} \\ &= 2v_W(\pi) + \sum_{m \in \{0, \dots, k-1\} \setminus \{0, l\}} (W_{m, i_{m+1} \bmod k} - W_{m, i_m}) \\ &\quad + W_{l, i_{l+1}} - W_{0, i_0} + W_{0, i_1} - W_{l, i_l} \\ &\quad (\text{This equality holds because } \alpha_0 = \alpha_l) \\ &= 2v_W(\pi) + \sum_{m=0}^{k-1} (W_{m, i_{m+1} \bmod k} - W_{m, i_m}) \\ &> 2v_W(\pi) \end{aligned}$$

Therefore, either  $v_W(P_1 \circ \pi) > v_W(\pi)$  or  $v_W(P_2 \circ \pi) > v_W(\pi)$ , which means that either  $P_1$  or  $P_2$  augments  $\pi$ . We reach a contradiction since  $P$  is assumed to be a shortest augmenting sequence of  $\pi$ . As a result, the  $k$  rows  $\alpha_0, \dots, \alpha_{k-1}$  are pairwise different. Because  $W$  has at most  $s$  distinct rows,  $k \leq s$ . The lemma holds.  $\square$

**Corollary 1.** *For any positive integers  $n$  and  $1 \leq r \leq n$ , any matrix  $W = AB$  with  $A \in D^{n \times r}$  and  $B \in \mathbb{R}^{r \times n}$  has locality at most  $|D|^r$ , where  $D \subseteq \mathbb{R}$  is a finite set.*

*Proof.* The matrix  $A$  has at most  $|D|^r$  distinct rows, meaning that so does  $W$ . The corollary follows immediately from Lemma 3.  $\square$

**Lemma 4.** *For any  $n$  and any matrix  $W \in [0, 1]^{n \times n}$  of rank  $r$ , there are matrices  $U \in [0, 1]^{n \times r}$  and  $V \in [-1, 1]^{r \times n}$  such that  $W = UV$ .*

*Proof.* Consider the linear subspace  $H$  spanned by all the column vectors  $\alpha_1, \dots, \alpha_n$  of  $W$ , i.e.  $H = \text{span}(\alpha_1, \dots, \alpha_n)$ . Define  $H^+ = H \cap (\mathbb{R}^+)^n$ , where  $\mathbb{R}^+$  stands for the set of non-negative real numbers.

The Euclidean distance in  $(\mathbb{R}^+)^n$  naturally induces a distance  $d_H$  in the subspace  $H$ . Hence, for a finite polyhedron  $P \subseteq H$ , we have a natural definition of the volume of  $P$  in  $H$ , denoted by  $\text{vol}_H(P)$ . Furthermore, for a vector  $v \in H^+$  and an  $(r-1)$ -dimensional linear subspace  $F \subset H$ , there is a natural concept of the distance from  $v$  to  $F$ , which is the Euclidean distance from the end point of  $v$  to  $F$ , denoted by  $d_H(v, F)$ .

In fact, there is a vector  $u \in H$ , denoted by  $u_H(F)$ , with 2-norm  $\|u\|_2 = 1$  and  $u \perp F$ , which is unique up to a scalar  $\pm 1$ . The distance  $d_H(v, F)$  is actually equal to  $|\langle u_H(F), v \rangle|$ , the absolute value of the standard Euclidean inner product of  $u_H(F)$  and  $v$ .

Given a set  $B = \{x_1, \dots, x_r\} \subseteq H^+$  of independent vectors, define the simplicial cone  $C(B)$  generated by  $B$  to be  $C(B) = \{\sum_{i=1}^r \lambda_i x_i \mid 0 \leq \lambda_i \text{ for each } i, \sum_{i=1}^r \lambda_i \leq 1\}$ .  $C(B)$  is a finite polyhedron in  $H$ , so its volume  $\text{vol}_H(C(B))$  is well defined. Consider, in the sense of infinity norm, the unit disk  $D_\infty = \{v \in \mathbb{R}^n : \|v\|_\infty \leq 1\}$ .

Arbitrarily choose an independent set  $A \subseteq D_\infty \cap H^+$  with the volume of its simplicial cone maximized, i.e.  $A = \text{argmax}_{B \subseteq (D_\infty \cap H^+), |B|=r} \text{vol}_H(C(B))$ . For each  $1 \leq i \leq r$ , let  $H_i = \text{span}(A \setminus \{x_i\})$  and  $H_i^+ = H_i \cap (\mathbb{R}^+)^n$ . Then  $C(A \setminus \{x_i\})$  is a simplicial cone in  $H_i^+$ . It is widely known in convex geometry

[2] that for the  $r$ -dimensional simplicial cone  $C(A)$ , the volume  $\text{vol}_H(C(A)) = \frac{1}{r} \text{vol}_{H_i}(C(A \setminus \{x_i\})) d_H(x_i, H_i)$ , for each  $1 \leq i \leq r$ .

Now, for an arbitrary vector  $y \in D_\infty \cap H^+$  and an arbitrary  $1 \leq i \leq r$ , we must have  $d_H(x_i, H_i) \geq d_H(y, H_i)$ . This follows from the three aspects:

- $\text{vol}_H(C(A)) = \frac{1}{r} \text{vol}_{H_i}(C(A \setminus \{x_i\})) d_H(x_i, H_i)$ .
- $\text{vol}_H(C(A \setminus \{x_i\} \cup \{y\})) = \frac{1}{r} \text{vol}_{H_i}(C(A \setminus \{x_i\})) d_H(y, H_i)$ .
- $\text{vol}_H(C(A)) \geq \text{vol}_H(C(A \setminus \{x_i\} \cup \{y\}))$  by the definition of  $A$ .

Now let  $u_i = u_H(H_i)$ , and assume that  $y = \lambda x_i + \beta$  with a unique  $\lambda \in \mathbb{R}$  and  $\beta \in H_i$ . Considering that

$$\begin{aligned} d_H(y, H_i) &= |\langle u_i, y \rangle| = |\langle u_i, \lambda x_i + \beta \rangle| \\ &= |\lambda \langle u_i, x_i \rangle + \langle u_i, \beta \rangle| = |\lambda| \langle u_i, x_i \rangle = |\lambda| d_H(x_i, H_i), \end{aligned}$$

then we have  $|\lambda| = \frac{d_H(y, H_i)}{d_H(x_i, H_i)} \leq 1$ . As a result, there exist  $\lambda_i \in [-1, 1]$ ,  $1 \leq i \leq r$ , such that  $y = \sum_{i=1}^r \lambda_i x_i$ .

Since  $W \in [0, 1]^{n \times n}$ , each column vector  $\alpha_j$ ,  $1 \leq j \leq n$ , satisfies  $\alpha_j \in D_\infty \cap H^+$ , implying that there is a vector  $\mu_j = (\mu_{1j}, \mu_{2j}, \dots, \mu_{rj})^T \in [-1, 1]^{r \times 1}$  such that  $\alpha_j = \sum_{i=1}^r x_i \mu_{ij}$ . As a result,  $W = (x_1, \dots, x_r)(\mu_1, \mu_2, \dots, \mu_n)$  with each  $x_i \in [0, 1]^{n \times 1}$  and each  $\mu_j \in [-1, 1]^{r \times 1}$ . The lemma thus holds.  $\square$

**Theorem 3.** *For every integer  $n > 0$ , every  $W \in [0, 1]^{n \times n}$ , and every  $\epsilon \in [0, 1]$ , there exists  $\widetilde{W} \in [0, 1]^{n \times n}$  such that  $\text{rank}(\widetilde{W}) \leq \text{rank}(W) + 1$ , the locality of  $\widetilde{W}$  is at most  $\lceil \frac{\text{rank}(W)}{\epsilon} \rceil^{\text{rank}(W)}$ , and  $\max_{1 \leq i, j \leq n} |W_{i,j} - \widetilde{W}_{i,j}| \leq \epsilon$ .*

*Proof.* Arbitrarily choose  $n > 0$ ,  $\epsilon \in [0, 1]$ , and  $W \in [0, 1]^{n \times n}$  with  $\text{rank}(W) = r$ . By Lemma 4, there are matrices  $U \in [0, 1]^{n \times r}$  and  $V \in [-1, 1]^{r \times n}$  such that  $W = UV$ .

Choose  $U' \in \{(\frac{1}{2} + k)\frac{\epsilon}{r} | k \in \mathbb{Z}^+, 0 \leq k < \lceil \frac{r}{\epsilon} \rceil\}^{n \times r}$  be such that  $|U'_{ij} - U_{ij}| \leq \frac{\epsilon}{2r}$  for all  $1 \leq i \leq n$  and  $1 \leq j \leq r$ , where  $\mathbb{Z}^+$  stands for the set of non-negative integers. Consider  $W' = U'V$ .  $W'$  has the following three properties:

1.  $\text{rank}(W') \leq r$ .
2.  $U'$  is defined on a finite domain of size  $\lceil \frac{r}{\epsilon} \rceil$ , so the locality of  $W'$  is at most  $\lceil \frac{r}{\epsilon} \rceil^r$  by Corollary 1.
3. For any  $1 \leq i, j \leq n$ ,  $|W'_{ij} - W_{ij}| = |\sum_{k=1}^r U'_{ik} V_{kj} - \sum_{k=1}^r U_{ik} V_{kj}| \leq \sum_{k=1}^r |U'_{ik} - U_{ik}| \leq \frac{\epsilon}{2}$ .

However, the domain of  $W'$  is  $[-\frac{\epsilon}{2}, 1 + \frac{\epsilon}{2}]$  rather than  $[0, 1]$ , by the third property. As a result, let  $\widetilde{W} = \frac{1}{1+\epsilon}(W' + \frac{\epsilon}{2}\mathbf{1})$ , where  $\mathbf{1}$  is the  $n$ -by- $n$  all-one matrix. Now we show that  $\widetilde{W}$  meets all the requirements of this lemma:

- One can immediately see that  $\text{rank}(\widetilde{W}) \leq \text{rank}(W') + 1 \leq r + 1$ .
- Since the domain of  $W'$  is  $[-\frac{\epsilon}{2}, 1 + \frac{\epsilon}{2}]$ , that of  $\widetilde{W}$  is  $[0, 1]$ .

- The locality of  $\widetilde{W}$  is exactly that of  $W'$ , so it is at most  $\lceil \frac{r}{\epsilon} \rceil^r$ .
- For any  $1 \leq i, j \leq n$ , since  $W_{ij} + \frac{\epsilon}{2} \geq W'_{ij}$ , we have  $W_{ij} + \epsilon \geq W'_{ij} + \frac{\epsilon}{2}$ . Then  $W_{ij} + \epsilon \geq \frac{1}{1+\epsilon}(W_{ij} + \epsilon) \geq \frac{1}{1+\epsilon}(W'_{ij} + \frac{\epsilon}{2}) = \widetilde{W}_{ij}$ . Likewise, one can show that  $\widetilde{W}_{ij} \geq W_{ij} - \epsilon$ .

Altogether, we have proven this theorem.  $\square$

## 4 Conclusion

This paper addresses the structure of perfect matchings of a weighted bipartite graph. It partially answers a 3-year question raised by David Karger. It shows that in general, the locality of a weighted bipartite graph can't be bounded by a universal function of the rank of the affiliated data. However, for any  $\epsilon > 0$ , there is an  $\epsilon$ -perturbation of the affiliated data such that the resulting locality is upper bounded by  $\lceil r/\epsilon \rceil^r$ , where  $r$  is the rank of the original data. This means that an arbitrary small perturbation can result in desired locality.

Furthermore, we show that if the weights are over a finite domain  $D$ , the locality of a weighted bipartite graph is indeed upper bounded by  $|D|^r$ , where  $r$  is the rank of the affiliated data. Though the bound is exponential of  $r$ , it is independent of the size of the graph.

For future work, we conjecture that in the finite domain case, the upper bound is polynomial of both  $|D|$  and  $r$ . And we will try smoothed analysis of the locality for general case.

## References

1. Arora, S.: Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM* 45(5), 753–782 (1998)
2. Ball, K.: An elementary introduction to modern convex geometry. In: *Flavors of Geometry*, pp. 1–58. Univ. Press (1997)
3. Duan, R., Pettie, S.: Approximating maximum weight matching in near-linear time. In: *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pp. 673–682 (2010)
4. Grötschel, M., Lovász, L., Schrijver, A.: *Geometric Algorithms and Combinatorial Optimization*. Springer (1988)
5. Karger, D.: Personal communication (2009)
6. Mitchell, J.S.B.: Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric tsp, k-mst, and related problems. *SIAM J. Comput.* 28(4), 1298–1309 (1999)
7. Papadimitriou, C.H., Steiglitz, K.: *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, Inc. (1982)
8. Sharathkumar, R., Agarwal, P.K.: A near-linear time  $\sqrt{\epsilon}$ -approximation algorithm for geometric bipartite matching. In: *Proceedings of the 44th Symposium on Theory of Computing*, pp. 385–394 (2012)
9. Vaidya, P.: Geometry helps in matching. In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pp. 422–425 (1988)
10. Vaidya, P.M.: Approximate minimum weight matching on points in k-dimensional space. *Algorithmica* 4(4), 569–583 (1989)

# Sublinear Time Approximate Sum via Uniform Random Sampling

Bin Fu<sup>1</sup>, Wenfeng Li<sup>2</sup>, and Zhiyong Peng<sup>2</sup>

<sup>1</sup> Department of Computer Science  
University of Texas-Pan American, Edinburg, TX 78539, USA  
[bfp@utpa.edu](mailto:bfp@utpa.edu)  
<sup>2</sup> Computer School  
Wuhan University, Wuhan, P.R. China  
[eyestar\\_2008@126.com](mailto:eyestar_2008@126.com), [peng@whu.edu.cn](mailto:peng@whu.edu.cn)

**Abstract.** We investigate the approximation for computing the sum  $a_1 + \dots + a_n$  with an input of a list of nonnegative elements  $a_1, \dots, a_n$ . If all elements are in the range  $[0, 1]$ , there is a randomized algorithm that can compute an  $(1 + \epsilon)$ -approximation for the sum problem in time  $O\left(\frac{n(\log \log n)}{\sum_{i=1}^n a_i}\right)$ , where  $\epsilon$  is a constant in  $(0, 1)$ . Our randomized algorithm is based on the uniform random sampling, which selects one element with equal probability from the input list each time. We also prove a lower bound  $\Omega\left(\frac{n}{\sum_{i=1}^n a_i}\right)$ , which almost matches the upper bound, for this problem.

**Keywords:** Randomization, Approximate Sum, Sublinear Time.

## 1 Introduction

Computing the sum of a list of elements has many applications. This problem can be found in the high school textbooks. In the textbook of calculus, we often see how to compute the sum of a list of elements, and decide if it converges when the number of items is infinite. Let  $\epsilon$  be a real number which is at least 0. A real number  $s$  is an  $(1 + \epsilon)$ -approximation for the sum problem  $a_1, a_2, \dots, a_n$  if  $\frac{\sum_{i=1}^n a_i}{1+\epsilon} \leq s \leq (1 + \epsilon) \sum_{i=1}^n a_i$ . When we have a huge number of data items and need to compute their sum, an efficient approximation algorithm becomes essential. Due to the fundamental importance of this problem, looking for a sublinear time solution for it is an interesting topic of research.

A similar problem is to compute the mean of a list of items  $a_1, a_2, \dots, a_n$ , whose mean is defined by  $\frac{a_1 + a_2 + \dots + a_n}{n}$ . Using  $O\left(\frac{1}{\epsilon^2} \log \frac{1}{\delta}\right)$  random samples, one can compute the  $(1 + \epsilon)$ -approximation for the mean, or decides if it is at most  $\delta$  [6]. In [3], Canetti, Even, and Goldreich showed that the sample size is tight. Dagum, Karp, Luby, and Ross [4] showed an algorithm to approximate the mean of a random variable in a time  $O(\rho/\mu^2)$ , where  $\rho = \max\{\sigma^2, \mu\}$  with variance  $\sigma$  and mean  $\mu$ . In [7], Motwani, Panigrahy, and Xu showed an  $O(\sqrt{n})$  time approximation scheme for computing the sum of  $n$  nonnegative elements. A

priority sampling approach for estimating subsets were studied in [1,5,2]. Using different cost and application models, they tried to build a sketch so that the sum of any subset can be computed approximately via the sketch.

We feel the uniform sampling is more justifiable than the weighted sampling. In this paper, we study the approximation for the sum problem under both deterministic model and randomized model. In the randomized model, we still use the uniform random samplings, and show how the time is  $O\left(\frac{n(\log \log n)}{\sum_{i=1}^n a_i}\right)$ . We also prove a lower bound that matches this time bound. An algorithm of time complexity  $O\left(\frac{n(\log \log n)}{\sum_{i=1}^n a_i}\right)$  for computing a list of nonnegative elements  $a_1, \dots, a_n$  in  $[0, 1]$  can be extended to a general list of nonnegative elements. It implies an algorithm of time complexity  $O\left(\frac{Mn \log \log n}{\sum_{i=1}^n a_i}\right)$  for computing a list of nonnegative elements of size at most  $M$  by converting each  $a_i$  into  $\frac{a_i}{M}$ , which is always in the range  $[0, 1]$ . Our randomized method, which is based on an interval partition of  $[0, 1]$ , is different from that used in [4].

## 2 Randomized Algorithm for the Sum Problem

In this section, we present a randomized algorithm for computing the approximate sum of a list of numbers in  $[0, 1]$ .

### 2.1 Chernoff Bounds

The analysis of our randomized algorithm often use the well known Chernoff bounds, which are described below. All proofs of this paper are self-contained except the following famous theorems in probability theory.

**Theorem 1** ([8]). *Let  $X_1, \dots, X_n$  be  $n$  independent random 0-1 variables, where  $X_i$  takes 1 with probability  $p_i$ . Let  $X = \sum_{i=1}^n X_i$ , and  $\mu = E[X]$ . Then for any  $\theta > 0$ ,*

1.  $\Pr(X < (1 - \theta)\mu) < e^{-\frac{1}{2}\mu\theta^2}$ , and
2.  $\Pr(X > (1 + \theta)\mu) < \left[\frac{e^\theta}{(1+\theta)^{(1+\theta)}}\right]^\mu$ .

We follow the proof of Theorem 1 to make the following versions (Theorem 3, and Theorem 2) of Chernoff bound for our algorithm analysis.

**Theorem 2.** *Let  $X_1, \dots, X_n$  be  $n$  independent random 0-1 variables, where  $X_i$  takes 1 with probability at least  $p$  for  $i = 1, \dots, n$ . Let  $X = \sum_{i=1}^n X_i$ , and  $\mu = E[X]$ . Then for any  $\theta > 0$ ,  $\Pr(X < (1 - \theta)pn) < e^{-\frac{1}{2}\theta^2 pn}$ .*

**Theorem 3.** *Let  $X_1, \dots, X_n$  be  $n$  independent random 0-1 variables, where  $X_i$  takes 1 with probability at most  $p$  for  $i = 1, \dots, n$ . Let  $X = \sum_{i=1}^n X_i$ . Then for any  $\theta > 0$ ,  $\Pr(X > (1 + \theta)pn) < \left[\frac{e^\theta}{(1+\theta)^{(1+\theta)}}\right]^{pn}$ .*

Define  $g_1(\theta) = e^{-\frac{1}{2}\theta^2}$  and  $g_2(\theta) = \frac{e^\theta}{(1+\theta)^{(1+\theta)}}$ . Define  $g(\theta) = \max(g_1(\theta), g_2(\theta))$ . We note that  $g_1(\theta)$  and  $g_2(\theta)$  are always strictly less than 1 for all  $\theta > 0$ . It is trivial for  $g_1(\theta)$ . For  $g_2(\theta)$ , this can be verified by checking that the function  $f(x) = x - (1+x) \ln(1+x)$  is decreasing and  $f(0) = 0$ . This is because  $f'(x) = -\ln(1+x)$  which is strictly less than 0 for all  $x > 0$ . Thus,  $g_2(\theta)$  is also decreasing, and less than 1 for all  $\theta > 0$ .

## 2.2 A Sublinear Time Algorithm

In this section, we show an algorithm to compute the approximate sum in sub-linear time in the cases that  $\sum_{i=1}^n a_i$  is at least  $(\log \log n)^{1+\epsilon}$  for any constant  $\epsilon > 0$ . This is a randomized algorithm with uniform random sampling.

**Theorem 4.** *Let  $\epsilon$  be a positive constant in  $(0, 1)$ . There is a sublinear time algorithm such that given a list of items  $a_1, a_2, \dots, a_n$  in  $[0, 1]$ , it gives a  $(1 + \epsilon)$ -approximation in time  $O(\frac{n(\log \log n)}{\sum_{i=1}^n a_i})$ .*

### Definition 1.

- For each interval  $I$  and a list of items  $L$ , define  $A(I, L)$  to be the number of items of  $L$  in  $I$ .
- For  $\delta$ , and  $\gamma$  in  $(0, 1)$ , a  $(\delta, \gamma)$ -partition for  $[0, 1]$  divides the interval  $[0, 1]$  into intervals  $I_1 = [\pi_1, \pi_0], I_2 = [\pi_2, \pi_1], I_3 = [\pi_3, \pi_2], \dots, I_k = [0, \pi_{k-1}]$  such that  $\pi_0 = 1, \pi_i = \pi_{i-1}(1 - \delta)$  for  $i = 1, 2, \dots, k - 1$ , and  $\pi_{k-1}$  is the first element  $\pi_{k-1} \leq \frac{\gamma}{n^2}$ .
- For a set  $A$ ,  $|A|$  is the number of elements in  $A$ . For a list  $L$  of items,  $|L|$  is the number of items in  $L$ .

A brief description of the idea is presented before the formal algorithm and its proof. In order to get an  $(1 + \epsilon)$ -approximation for the sum of  $n$  input numbers in the list  $L$ , a parameter  $\delta$  is selected with  $1 - \frac{\epsilon}{2} \leq (1 - \delta)^3$ . For a  $(\delta, \delta)$ -partition  $I_1 \cup I_2 \dots \cup I_k$  for  $[0, 1]$ , Algorithm Approximate-Sum(.) below gives the estimation for the number of items in each  $I_j$  if interval  $I_j$  has a sufficient number of items. Otherwise, those items in  $I_j$  can be ignored without affecting much of the approximation ratio. We have an adaptive way to do random samplings in a series of phases. Let  $s_t$  denote the number of random samples in phase  $t$ . Phase  $t + 1$  doubles the number of random samples of phase  $t$  ( $s_{t+1} = 2s_t$ ). Let  $L$  be the input list of items in the range  $[0, 1]$ . Let  $d_j$  be the number items in  $I_j$  from the samples. For each phase, if an interval  $I_j$  shows sufficient number of items from the random samples, the number of items  $A(I_j, L)$  in  $I_j$  can be sufficiently approximated by  $\hat{A}(I_j, L) = d_j \cdot \frac{n}{s_t}$ . Thus,  $\hat{A}(I_j, L)\pi_j$  also gives an approximation for the sum of the sizes of items in  $I_j$ . The sum  $\text{apx\_sum} = \sum_{I_j} \hat{A}(I_j, L)\pi_j$  for those intervals  $I_j$  with a large number of samples gives an approximation for the total sum  $\sum_{i=1}^n a_i$  of the input list. In the early stages,  $\text{apx\_sum}$  is much smaller than  $\frac{n}{s_t}$ . Eventually,  $\text{apx\_sum}$  will surpass  $\frac{n}{s_t}$ . This happens when  $s_t$  is more than  $\frac{n}{\sum_{i=1}^n a_i}$  and  $\text{apx\_sum}$  is close to the sum  $\sum_{i=1}^n a_i$  of all items from the input list. This indicates that

the number of random samples is sufficient for our approximation algorithm. For those intervals with small number of samples, their items only form a small fraction of the total sum. This process is terminated when ignoring all those intervals with none or small number of samples does not affect much of the accuracy of approximation. The algorithm gives up the process of random sampling when  $s_t$  surpasses  $n$ , and switches to a deterministic way to access the input list, which happens when the total sum of the sizes of input items is  $O(1)$ .

The computation time at each phase  $i$  is  $O(s_i)$ . If phase  $t$  is the last phase, the total time is  $O(s_t + \frac{s_t}{2} + \frac{s_t}{2^2} + \dots) = O(s_t)$ , which is close to  $O(\frac{n}{\sum_{i=1}^n a_i})$ . Our final complexity upper bound is  $O(\frac{n(\log \log n)}{\sum_{i=1}^n a_i})$ , where  $\log \log n$  factor is caused by the probability amplification of  $O(\log n)$  stages and  $O(\log n)$  intervals of the  $(\delta, \delta)$  partition in the randomized algorithm.

### Algorithm Approximate-Sum( $\epsilon, \alpha, n, L$ )

Input: a parameter, a small parameter  $\epsilon \in (0, 1)$ , a failure probability upper bound  $\alpha$ , an integer  $n$ , a list  $L$  of  $n$  items  $a_1, \dots, a_n$  in  $[0, 1]$ .

Steps:

1. Phase 0:
  2. Select  $\delta = \frac{\epsilon}{6}$  that satisfies  $1 - \frac{\epsilon}{2} \leq (1 - \delta)^3$ .
  3. Let  $P$  be a  $(\delta, \delta)$ -partition  $I_1 \cup I_2 \dots \cup I_k$  for  $[0, 1]$ .
  4. Let  $\xi_0$  be a parameter such that  $8(k+1)(\log n)g(\delta)^{(\xi_0 \log \log n)/2} < \alpha$  for all large  $n$ .
  5. Let  $z := \xi_0 \log \log n$ .
  6. Let parameters  $c_1 := \frac{\delta^2}{2(1+\delta)}$ , and  $c_2 := \frac{12\xi_0}{(1-\delta)c_1}$ .
  7. Let  $s_0 := z$ .
  8. End of Phase 0.
9. Phase  $t$ :
  10. Let  $s_t := 2s_{t-1}$ .
  11. Sample  $s_t$  random items  $a_{i_1}, \dots, a_{i_{s_t}}$  from the input list  $L$ .
  12. Let  $d_j := |\{h : a_{i_h} \in I_j \text{ and } 1 \leq h \leq s_t\}|$  for  $j = 1, 2, \dots, k$ .
  13. For each  $I_j$ ,
    14. if  $d_j \geq z$ ,
    15. then let  $\hat{A}(I_j, L) := \frac{n}{s_t}d_j$  to approximate  $A(I_j, L)$ .
    16. else let  $\hat{A}(I_j, L) := 0$ .
  17. Let  $\text{apx\_sum} := \sum_{d_j \geq z} \hat{A}(I_j, L)\pi_j$  to approximate  $\sum_{i=1}^n a_i$ .
  18. If  $\text{apx\_sum} \leq \frac{2c_2 n \log \log n}{s_t}$  and  $s_t < n$  then enter Phase  $t + 1$ .
  19. else
    20. If  $s_t < n$
    21. then let  $\text{apx\_sum} := \sum_{d_j \geq z} \hat{A}(I_j, L)\pi_j$  to approximate  $\sum_{1 \leq i \leq n} a_i$ .
    22. else let  $\text{apx\_sum} := \sum_{i=1}^n a_i$ .
    23. Output  $\text{apx\_sum}$  and terminate the algorithm.
  24. End of Phase  $t$ .

### End of Algorithm

Several lemmas will be proved in order to show the performance of the algorithm. Let  $\delta, \xi_0, c_1$ , and  $c_2$  be parameters defined as those in the Phase 0 of the algorithm Approximate-Sum(.)

**Lemma 1.**

1. For parameter  $\delta$  in  $(0, 1)$ , a  $(\delta, \delta)$ -partition for  $[0, 1]$  has the number of intervals  $k = O(\frac{\log n + \log \frac{1}{\delta}}{\delta})$ .
2.  $g(x) \leq e^{-\frac{x^2}{4}}$  when  $0 < x \leq \frac{1}{2}$ .
3. The parameter  $\xi_0$  can be set to be  $O(\frac{\log \frac{1}{\alpha\delta}}{\log \frac{1}{g(\delta)}}) = O(\frac{\log \frac{1}{\alpha\delta}}{\delta^2})$  for line 4 in the algorithm Approximate-Sum(.)
4. Function  $g(x)$  is decreasing and  $g(x) < 1$  for every  $x > 0$ .

*Proof.* Statement 1: The number of intervals  $k$  is the least integer with  $(1 - \delta)^k \leq \frac{\delta}{n^2}$ . We have  $k = O(\frac{\log n + \log \frac{1}{\delta}}{\delta})$ .

Statement 2: By definition  $g(x) = \max(g_1(x), g_2(x))$ , where  $g_1(x) = e^{-\frac{1}{2}x^2}$  and  $g_2(x) = \frac{e^x}{(1+x)^{(1+x)}}$ . We just need to prove that  $g_2(x) \leq e^{-\frac{x^2}{4}}$  when  $x \leq \frac{1}{2}$ . By Taylor theorem  $\ln(1+x) \geq x - \frac{x^2}{2}$ . Assume  $0 < x \leq \frac{1}{2}$ . We have

$$\ln g_2(x) = x - (1+x) \ln(1+x) \leq x - (1+x)(x - \frac{x^2}{2}) = -\frac{x^2}{2}(1-x) \leq -\frac{x^2}{4}.$$

Statement 3: We need to set up  $\xi_0$  to satisfy the condition in line 4 in the algorithm. It follows from statement 1 and statement 2.

Statement 4: It follows from the fact that  $g_2(x)$  is decreasing, and less than 1 for each  $x > 0$ . We already explained in section 2.1.

We use the uniform random sampling to approximate the number of items in each interval  $I_j$  in the  $(\delta, \delta)$ -partition. Due to the technical reason, we estimate the failure probability instead of the success probability.

**Lemma 2.** Let  $Q_1$  be the probability that the following statement is false at the end of each phase:

(i) For each interval  $I_j$  with  $d_j \geq z$ ,  $(1 - \delta)A(I_j, L) \leq \hat{A}(I_j, L) \leq (1 + \delta)A(I_j, L)$ .

Then for each phase in the algorithm,  $Q_1 \leq (k+1) \cdot g(\delta)^{\frac{z}{2}}$ .

*Proof.* An element of  $L$  in  $I_j$  is sampled (by an uniform sampling) with probability  $p_j = \frac{A(I_j, L)}{n}$ . Let  $p' = \frac{z}{2s_t}$ . For each interval  $I_j$  with  $d_j \geq z$ , we discuss two cases.

– Case 1.  $p' \geq p_j$ .

In this case,  $d_j \geq z \geq 2p's_t \geq 2p_j s_t$ . Note that  $d_j$  is the number of elements in interval  $I_j$  among  $s_t$  random samples  $a_{i_1}, \dots, a_{i_{s_t}}$  from  $L$ . By Theorem 3 (with  $\theta = 1$ ), with probability at most  $P_1 = g_2(1)^{p_j m_t} \leq g_2(1)^{p's_t} \leq g_2(1)^{z/2} \leq g(1)^{z/2}$ , there are at least  $2p_j s_t$  samples are from interval  $I_j$ . Thus, the probability is at most  $P_1$  for the condition of Case 1 to be true.

– Case 2.  $p' < p_j$ .

By Theorem 3, we have  $\Pr[d_j > (1 + \delta)p_j m_t] \leq g_2(\delta)^{p_j m_t} \leq g_2(\delta)^{p' s_t} \leq g_2(\delta)^{\frac{z}{2}} \leq g(\delta)^{\frac{z}{2}}$ .

By Theorem 2, we have  $\Pr[d_j \leq (1 - \delta)p_j m_t] \leq g_1(\delta)^{p_j m_t} \leq g_1(\delta)^{p' s_t} = g_1(\delta)^{\frac{z}{2}} \leq g(\delta)^{\frac{z}{2}}$ .

For each interval  $I_j$  with  $d_j \geq z$  and  $(1 - \delta)p_j m_t \leq d_j \leq (1 + \delta)p_j m_t$ , we have  $(1 - \delta)A(I_j, L) \leq \hat{A}(I_j, L) \leq (1 + \delta)A(I_j, L)$  by line 15 in Approximate-Sum(.)

There are  $k$  intervals  $I_1, \dots, I_k$ . Therefore, with probability at most  $P_2 = k \cdot g(\delta)^{\frac{z}{2}}$ , the following is false: For each interval  $I_j$  with  $d_j \geq z$ ,  $(1 - \delta)A(I_j, L) \leq \hat{A}(I_j, L) \leq (1 + \delta)A(I_j, L)$ .

By the analysis of Case 1 and Case 2, we have  $Q_1 \leq P_1 + P_2 \leq (k + 1) \cdot g(\delta)^{\frac{z}{2}}$  (see statement 4 of Lemma 1). Thus, the lemma has been proven.

**Lemma 3.** Assume that  $s_t \geq \frac{c_2 n \log \log n}{\sum_{i=1}^n a_i}$ . Then right after executing Phase t in Approximate-Sum(.), with probability at most  $Q_2 = 2kg(\delta)^{\xi_0 \log \log n}$ , the following statement is false:

(ii) For each interval  $I_j$  with  $A(I_j, L) \geq c_1 \sum_{i=1}^n a_i$ , A).  $(1 - \delta)A(I_j, L) \leq \hat{A}(I_j, L) \leq (1 + \delta)A(I_j, L)$ ; and B).  $d_j \geq z$ .

*Proof.* Assume that  $s_t \geq \frac{c_2 n \log \log n}{\sum_{i=1}^n a_i}$ . Consider each interval  $I_j$  with  $A(I_j, L) \geq c_1 \sum_{i=1}^n a_i$ . We have that  $p_j = \frac{A(I_j, L)}{n} \geq \frac{c_1 \sum_{i=1}^n a_i}{n}$ . An element of  $L$  in  $I_j$  is sampled with probability  $p_j$ . By Theorem 3, Theorem 2, and Phase 0 of Approximate-Sum(.), we have

$$\Pr[d_j < (1 - \delta)p_j m_t] \leq g_1(\delta)^{p_j m_t} \leq g_1(\delta)^{c_1 c_2 \log \log n} \leq g(\delta)^{\xi_0 \log \log n}. \quad (1)$$

$$\Pr[d_j > (1 + \delta)p_j m_t] \leq g_2(\delta)^{p_j m_t} \leq g_2(\delta)^{c_1 c_2 \log \log n} \leq g(\delta)^{\xi_0 \log \log n}. \quad (2)$$

Therefore, with probability at most  $2kg(\delta)^{\xi_0 \log \log n}$ , the following statement is false:

For each interval  $I_j$  with  $A(I_j, L) \geq c_1 \sum_{i=1}^n a_i$ ,  $(1 - \delta)A(I_j, L) \leq \hat{A}(I_j, L) \leq (1 + \delta)A(I_j, L)$ .

If  $d_j \geq (1 - \delta)p_j s_t$ , then we have

$$\begin{aligned} d_j &\geq (1 - \delta) \frac{A(I_j, L)}{n} s_t \geq (1 - \delta) \frac{(c_1 \sum_{i=1}^n a_i)}{n} \cdot \frac{c_2 n \log \log n}{\sum_{i=1}^n a_i} = (1 - \delta) c_1 c_2 \log \log n \\ &\geq \xi_0 \log \log n = z. \quad (\text{by Phase 0 of Approximate-Sum(.)}) \end{aligned}$$

**Lemma 4.** The total sum of the sizes of items in those  $I_j$ 's with  $A(I_j, L) < c_1 \sum_{i=1}^n a_i$  is at most  $\frac{\delta}{2}(\sum_{i=1}^n a_i) + \frac{\delta}{n}$ .

*Proof.* By Definition 1, we have  $\pi_j = (1 - \delta)^j$  for  $j = 1, \dots, k - 1$ . We have that

- the sum of sizes of items in  $I_k$  is at most  $n \cdot \frac{\delta}{n^2} = \frac{\delta}{n}$ ,
- for each interval  $I_j$  with  $A(I_j, L) < c_1 \sum_{i=1}^n a_i$ , the sum of sizes of items in  $I_j$  is at most  $(c_1 \sum_{i=1}^n a_i) \pi_{j-1} \leq (c_1 \sum_{i=1}^n a_i)(1 - \delta)^{j-1}$  for  $j \in [1, k - 1]$ .

The total sum of the sizes of items in those  $I_j$ s with  $A(I_j, L) < c_1 \sum_{i=1}^n a_i$  is at most

$$\begin{aligned} & \sum_{j=1}^{k-1} (c_1 \sum_{i=1}^n a_i) \pi_{j-1} + \sum_{a_i \in I_k} a_k \leq \sum_{j=1}^{k-1} (c_1 \sum_{i=1}^n a_i) (1 - \delta)^{j-1} + n \cdot \frac{r}{n^2} \\ & \leq \frac{c_1}{\delta} (\sum_{i=1}^n a_i) + \frac{\delta}{n} \leq \frac{\delta}{2} (\sum_{i=1}^n a_i) + \frac{\delta}{n}. \quad (\text{by Phase 0 of Approximate-Sum(.)}) \end{aligned}$$

**Lemma 5.** Assume that at the end of phase  $t$ , for each  $I_j$  with  $\hat{A}(I_j, L) > 0$ ,  $A(I_j, L)(1 - \delta) \leq \hat{A}(I_j, L) \leq A(I_j, L)(1 + \delta)$ ; and  $d_j \geq z$  if  $A(I_j, L) \geq c_1 \sum_{i=1}^n a_i$ . Then  $(1 - \frac{\epsilon}{2})(\sum_{i=1}^n a_i - \frac{4\delta}{n}) \leq \text{apx\_sum} \leq (1 + \delta)(\sum_{i=1}^n a_i)$  at the end of phase  $t$ .

**Lemma 6.** With probability at most  $Q_5 = (k+1) \cdot (\log n)g(\delta)^{\frac{z}{2}}$ , at least one of the following statements is false:

- A. For each phase  $t$  with  $s_t < \frac{c_2 n \log \log n}{\sum_{i=1}^n a_i}$ , the condition  $\text{apx\_sum} \leq \frac{2c_2 n \log \log n}{s_t}$  in line 18 of the algorithm is true.
- B. If  $\sum_{i=1}^n a_i \geq 4$ , then the algorithm stops some phase  $t$  with  $s_t \leq \frac{16c_2 n \log \log n}{\sum_{i=1}^n a_i}$ .
- C. If  $\sum_{i=1}^n a_i < 4$ , then it stops at a phase  $t$  in which the condition  $s_t \geq n$  first becomes true, and outputs  $\text{apx\_sum} = \sum_{i=1}^n a_i$ .

**Lemma 7.** The complexity of the algorithm is  $O(\frac{\log \frac{1}{\delta}}{\delta^4} \min(\frac{n}{\sum_{i=1}^n a_i}, n) \log \log n)$ . In particular, the complexity is  $O(\min(\frac{n}{\sum_{i=1}^n a_i}, n) \log \log n)$  if  $\alpha$  is fixed in  $(0, 1)$ .

**Lemma 8.** With probability at most  $\alpha$ , at least one of the following statements is false after executing the algorithm Approximate-Sum( $\epsilon, \alpha, n, L$ ):

- 1. If  $\sum_{i=1}^n a_i \geq 4$ , then  $(1 - \epsilon)(\sum_{i=1}^n a_i) \leq \text{apx\_sum} \leq (1 + \frac{\epsilon}{2})(\sum_{i=1}^n a_i)$ ;
- 2. If  $\sum_{i=1}^n a_i < 4$ , then  $\text{apx\_sum} = \sum_{i=1}^n a_i$ ; and
- 3. It runs in  $O(\frac{\log \frac{1}{\delta}}{\delta^4} \min(\frac{n}{\sum_{i=1}^n a_i}, n) \log \log n)$  time. In particular, the complexity of the algorithm is  $O(\min(\frac{n}{\sum_{i=1}^n a_i}, n) \log \log n)$  if  $\alpha$  is fixed in  $(0, 1)$ .

Now we give the proof for our main theorem.

*Proof (for Theorem 4).* Let  $\alpha = \frac{1}{4}$  and  $\epsilon \in (0, 1)$ . It follows from Lemma 8 via a proper setting for those parameters in the algorithm Approximate-Sum(.)

The  $(\delta, \delta)$ -partition  $P : I_1 \cup I_2 \dots \cup I_k$  for  $[0, 1]$  can be generated in  $O(\frac{\log n + \log \frac{1}{\delta}}{\delta})$  time by Lemma 1. Let  $L$  be a list of  $n$  numbers in  $[0, 1]$ . Pass  $\delta, \alpha, P, n$ , and  $L$  to Approximate-Sum(.), which returns an approximate sum  $\text{apx\_sum}$ .

By statement 1 and statement 2 of Lemma 8, we have an  $(1 + \epsilon)$ -approximation for the sum problem with failure probability at most  $\alpha$ . The computational time is bounded by  $O(\frac{\log \frac{1}{\delta}}{\delta^4} \min(\frac{n}{\sum_{i=1}^n a_i}, n) \log \log n)$  by statement 3 of Lemma 8.

**Definition 2.** Let  $f(n)$  be a function from  $N$  to  $(0, +\infty)$  with  $f(n) \leq n$  and a parameter  $c > 1$ . Define  $\sum(c, f(n))$  be the class of sum problem with an input of nonnegative numbers  $a_1, \dots, a_n$  in  $[0, a]$  with  $\sum_{i=1}^n a_i \in [\frac{f(n)}{c}, cf(n)]$ .

**Corollary 1.** Assume that  $f(n)$  is a function from  $N$  to  $(0, +\infty)$  with  $f(n) \leq n$ , and  $c$  is a given constant  $c$  greater than 1. There is a  $O(\frac{n(\log \log n)}{f(n)})$  time algorithm such that given a list of nonnegative numbers  $a_1, a_2, \dots, a_n$  in  $\sum(c, f(n))$ , it gives a  $(1 - \epsilon)$ -approximation.

### 3 Lower Bound

We show a lower bound for those sum problems with bounded sum of sizes  $\sum_{i=1}^n a_i$ . The lower bound almost matches the upper bound.

**Theorem 5.** Assume  $f(n)$  is an nondecreasing unbounded function from  $N$  to  $(0, +\infty)$  with  $f(n) \leq n$  and  $f(n) = o(n)$ . Every randomized  $(\sqrt{c} - \epsilon)$ -approximation algorithm for the sum problem in  $\sum(c, f(n))$  (see Definition 2) needs  $\Omega(\frac{n}{f(n)})$  time, where  $c$  is a constant greater than 1, and  $\epsilon$  is an arbitrary small constant in  $(0, \sqrt{c} - 1)$ .

**Acknowledgments.** The authors are also grateful to anonymous referees for providing comments to help us improve the presentation of this paper, and pointing the reference [4], which is related to our work.

### References

1. Alon, N., Duffield, N., Lund, C., Thorup, M.: Estimating arbitrary subset sums with few probes. In: Proc. PODS, pp. 317–325 (2005)
2. Broder, A., Fontura, M., Josifovski, V., Kumar, R., Motwani, R., Nabar, S., Panigrahy, R., Tomkins, A., Xu, Y.: Estimating corpus size via queries. In: Proceedings of the 15th ACM International Conference on Information and Knowledge Management (CIKM 2006), pp. 594–603 (2006)
3. Canetti, R., Even, G., Goldreich, O.: Lower bounds for sampling algorithms for estimating the average. Information Processing Letters 53, 17–25 (1995)
4. Dagum, P., Karp, R., Luby, M., Ross, S.: An optimal algorithm for monte carlo estimation. SIAM J. Comput. 29(5), 1484–1496 (2000)
5. Duffield, N., Lund, C., Thorup, M.: Learn more, sample less: control of volume and variance in network measurements. IEEE Trans. on Information Theory 51, 1756–1775 (2005)
6. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association 58, 13–30 (1963)
7. Motwani, R., Panigrahy, R., Xu, Y.: Estimating sum by weighted sampling. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 53–64. Springer, Heidelberg (2007)
8. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press (2000)

# Tractable Connected Domination for Restricted Bipartite Graphs (Extended Abstract)<sup>\*</sup>

Zhao Lu<sup>1</sup>, Tian Liu<sup>1</sup>, and Ke Xu<sup>2</sup>

<sup>1</sup> Key Laboratory of High Confidence Software Technologies, Ministry of Education,  
Peking University, Beijing 100871, China  
[lt@pku.edu.cn](mailto:lt@pku.edu.cn)

<sup>2</sup> National Lab of Software Development Environment,  
Beihang University, Beijing 100191, China  
[kexu@nlsde.buaa.edu.cn](mailto:kexu@nlsde.buaa.edu.cn)

**Abstract.** Finding a minimum connected dominating set (*connected domination*) is known  $\mathcal{NP}$ -complete for chordal bipartite graphs, but tractable for convex bipartite graphs. In this paper, connected domination is shown tractable for circular- and triad-convex bipartite graphs, by efficient reductions from these graphs to convex bipartite graphs.

**Keywords:** Connected domination, polynomial-time, circular-convex bipartite graph, triad-convex bipartite graph, convex bipartite graph.

## 1 Introduction

A *connected dominating set* in a graph is a subset of vertices, which induces a connected subgraph and every vertex outside it has a neighbor in it. The problem of finding a minimum connected dominating set, called *connected domination* [10], are known  $\mathcal{NP}$ -complete for general graphs [2], bipartite graphs [9], chordal bipartite graphs [8], and tractable for convex bipartite graphs [1].

In a *convex bipartite* [3] (*circular-convex bipartite* [7], *tree-convex bipartite* [4,5], respectively) graph, there is a linear ordering (circular ordering, tree, respectively) defined on one class of the vertices, such that for every vertex in another class, the neighborhood of this vertex is an interval (a circular arc, a subtree, respectively). When the tree is a *triad*, i.e. three paths with a common end, the graph is called *triad-convex bipartite* [6,11,5].

An interesting theoretical problem is

- What is the boundary between tractability and intractability of connected domination for bipartite graphs?

In this paper, we make partial progress on this problem by showing that connected domination is tractable for circular- and triad-convex bipartite graphs. Our results extend the known tractability of connected domination from convex bipartite graphs to circular- and triad-convex bipartite graphs.

---

\* Partially supported by National 973 Program of China (Grant No. 2010CB328103).

Before our work, the complexity results for circular- and triad-convex bipartite graphs are scarce [7,6,11,5]. We make polynomial time Turing reductions (i.e. Cook reductions [2]) of connected domination from circular- and triad-convex bipartite graphs to convex bipartite graphs. This method may be of use to show more problems tractable for circular- and triad-convex bipartite graphs.

This paper is structured as follows. In next section, necessary definitions and properties are given. In last two sections, the reductions from circular- and triad-convex bipartite graphs to convex bipartite graphs are given, respectively.

## 2 Preliminaries

In a graph  $G = (V, E)$ , the neighborhood of a vertex  $x \in V$  is  $N(x) = \{v \mid v \text{ is adjacent to } x\}$ , and the closed neighborhood of  $x$  is  $N[x] = N(x) \cup \{x\}$ . For a subset  $D \subseteq V$ ,  $N(D)$  is the union of all  $N(x)$  with  $x \in D$ , and  $N[D] = N(D) \cup D$ . A set  $X$  is said to be *dominated* by  $D$ , if  $X \subseteq N[D]$ . If  $X = \{x\}$ , then vertex  $x$  is dominated by  $D$ . If  $V$  is dominated by  $D$ ,  $D$  is called a *dominating set*. A dominating set  $D$  is called connected if  $D$  induces a connected subgraph. In general, if the induced subgraph of a vertex set is connected, we say that the vertex set is connected. The problem of finding a minimum connected dominating set in a graph is called *connected domination*. In this paper,  $D$  will denote a (minimum) connected dominating set of  $G$ . The following properties of connected dominating set is useful. If  $D$  is a connected dominating set in graph  $G$ , then for every vertex  $x$  in  $G$ , we have  $N(x) \cap D \neq \emptyset$ .

For a bipartite graph  $G = (A, B, E)$  and two vertex subset  $A' \subseteq A$  and  $B' \subseteq B$ , we use  $G[A', B']$  to denote the subgraph induced by  $A'$  and  $B'$ .

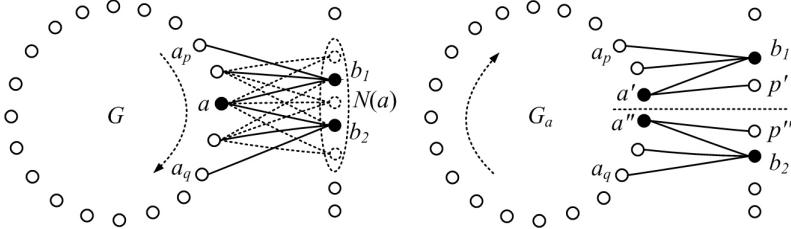
## 3 Reduction for Circular-Convex Bipartite Graphs

For circular-convex bipartite graph  $G = (A, B, E)$ , we always assume a circular ordering  $<$  on  $A$ ; for every vertex in  $B$ , its neighborhood is a circular arc on  $A$ .

**Lemma 1.** *Assume that  $G = (A, B, E)$  is a circular-convex bipartite graph, with a circular ordering on  $A$ , and  $D$  is a minimum connected dominating set of  $G$ . Then for every vertex  $a$  in  $D \cap A$ , it holds that  $|D \cap N(a)| \leq 2$ .*

*Proof.* Since  $G$  is circular-convex bipartite,  $N(N(a))$  is a circular arc in  $A$ . Let the two ends of this arc be  $a_p$  and  $a_q$ , respectively. Then there are two vertices  $b_1$  and  $b_2$  in  $N(a)$ , such that  $a_p \in N(b_1)$  and  $a_q \in N(b_2)$ . Then,  $N(N(a)) = \{x \mid a_p \leq x \leq a_q\}$ , see Figure 1 (left). Since  $a_p, \dots, a \in N(b_1)$  and  $a, \dots, a_q \in N(b_2)$ , we have that  $N(b_1) \cup N(b_2) = N(N(a))$ . Thus  $\{a, b_1, b_2\}$  is connected and dominates  $N(N(a)) \cup N(a)$ . For any connected dominating set  $D'$  which contains  $a$ ,  $D' \setminus N(a) \cup \{b_1, b_2\}$  is still a connected dominating set. So if  $|D' \cap N(a)| > 2$ , then  $D'$  is not minimum.  $\square$

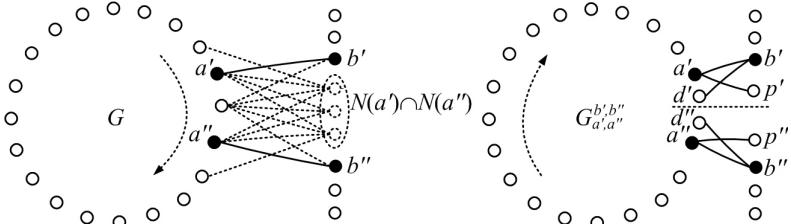
For a minimum connected dominating set  $D$  of  $G$ , we consider the following two cases.



**Fig. 1.** Graph  $G$  (left) and graph  $G_a$  (right)

**Case 1.** There is a circle  $a_{i_1}b_{i_1} \cdots a_{i_t}b_{i_t}a_{i_1}$  in  $D$  such that  $a_{i_1} < \cdots < a_{i_t}$ . By Lemma 1,  $N(a_{i_j}) \cap D = \{b_{i_{j-1}}, b_{i_j}\}$  for  $1 \leq j \leq t$ , where  $b_{i_0} = b_{i_t}$ . Thus, for any  $a_{i_j}$ ,  $D \setminus \{a_{i_j}\}$  is still connected. Then, for any  $a \in A$ , we define a graph  $G_a$  as follows. Let  $b_1, b_2, b_p, b_q$  be as in proof of Lemma 1. The graph  $G_a$  is defined by removing vertices in  $N(a) \setminus \{b_1, b_2\}$ , restricting  $b_1, b_2$  adjacent to only one side of  $a$  respectively by splitting vertex  $a$  into  $a', a''$ , and adding two vertex  $p', p''$  with  $p'$  only adjacent to  $a'$  and  $p''$  only adjacent to  $a''$ . That is,  $G_a = (A_a, B_a, E_a)$ , where  $A_a = A \setminus \{a\} \cup \{a', a''\}$ ,  $B_a = B \setminus N(a) \cup \{b_1, b_2, p', p''\}$ , and  $E_a = E \setminus \{e \in E \mid e \text{ is incident to a vertex in } N[a]\} \cup \{(a', b_1), (a', p'), (a'', p''), (a'', b_2)\} \cup \{(a_x, b_1) \in E \mid a_p \leq a_x < a\} \cup \{(a_y, b_2) \in E \mid a < a_y \leq a_q\}$ . See Figure 1.

*Remark 1.* In definition of  $G_a$ , we assume that  $|D \cap N(a)| = 2$  and  $a \notin \{a_p, a_q\}$ . If  $|D \cap N(a)| = 1$  and  $a \notin \{a_p, a_q\}$ , then just thinking  $b_1$  and  $b_2$  is resulted by splitting the unique  $b \in D \cap N(a)$ . If  $a \in \{a_p, a_q\}$ , then  $G$  is convex bipartite. Adding  $p', p''$  is to force  $a', a''$  into every connected dominating set of  $G_a$ .



**Fig. 2.** Graph  $G$  (left) and graph  $G_{a',a''}^{b',b''}$  (right)

**Case 2.** Otherwise, there are  $a', a'' \in D$ , such that  $D \cap N(a') \cap N(a'') = \emptyset$  and  $D \cap \{a \in A \mid a' < a < a''\} = \emptyset$ . See Figure 2 (left). Then, there are  $b' \in D \cap (N(a') \setminus N(a''))$  and  $b'' \in D \cap (N(a'') \setminus N(a'))$ , such that  $\{a \in A \mid a' < a < a''\} \subseteq N(b') \cup N(b'')$ . For any  $a', a'' \in A$ , if there are  $b' \in N(a') \setminus N(a'')$  and  $b'' \in N(a'') \setminus N(a')$ , such that  $\{a \in A \mid a' < a < a''\} \subseteq N(b') \cup N(b'')$ , then call such a quadruple  $(a', a'', b', b'')$  *good*, and the associated dominating set as above is also called *good*. For any good quadruple  $(a', a'', b', b'')$ , we define a

graph  $G_{a',a''}^{b',b''}$  by removing vertices in  $N(a') \cap N(a'')$  and  $\{a \in A | a' < a < a''\}$ , adding four vertices  $d', d'', p', p''$  such that  $d'$  only adjacent to  $b'$ ,  $d''$  only adjacent to  $b''$ ,  $p'$  only adjacent to  $a'$ , and  $p''$  only adjacent to  $a''$ , respectively. That is,  $G_{a',a''}^{b',b''} = (A_{a',a''}^{b',b''}, B_{a',a''}^{b',b''}, E_{a',a''}^{b',b''})$ , where  $A_{a',a''}^{b',b''} = A \setminus \{a \in A | a' < a < a''\} \cup \{d', d''\}$ ,  $B_{a',a''}^{b',b''} = B \setminus (N(a') \cap N(a'')) \cup \{p', p''\}$ , and  $E_{a',a''}^{b',b''} = E \setminus \{e \in E | e \text{ has one end in } A_{a',a''}^{b',b''} \text{ and other end in } B_{a',a''}^{b',b''}\} \cup \{(a', p'), (a'', p''), (d', b'), (d'', b'')\}$ . See Figure 2.

**Remark 2.** In definition of  $G_{a',a''}^{b',b''}$ , adding  $b', b'', p', p''$  is to force  $a', a'', b', b''$  into every connected dominating set of  $G_{a',a''}^{b',b''}$ . Note that for a good quadruple  $(a', a'', b', b'')$ , every connected dominating set of  $G_{a',a''}^{b',b''}$  is also a connected dominating set of  $G$ .

Due to space limitation, we omit detail in proofs about properties on  $G_{a',a''}^{b',b''}$ .

**Lemma 2.** For any  $a \in A$ ,  $G_a$  is convex bipartite. Moreover, for any good quadruple  $(a', a'', b', b'')$ ,  $G_{a',a''}^{b',b''}$  is convex bipartite.

*Proof.* Since all vertices in  $N(a) \setminus \{b_1, b_2\}$  are removed, and  $b_1$  is only adjacent to  $a_p, \dots, a'$  and  $b_2$  is only adjacent to  $a'', \dots, a_q$ , no vertex in  $B_a$  is adjacent to both  $a'$  and  $a''$ , and for each vertex in  $B_a$ , its neighborhood is an interval under the linear ordering  $a'' < \dots < a_q < \dots < a_p < \dots < a'$ , see Figure 1 (right). Thus  $G_a$  is convex bipartite. Similarly,  $G_{a',a''}^{b',b''}$  is convex bipartite.  $\square$

**Lemma 3.** For any  $a \in A$  and a connected dominating set  $D$  of  $G$  containing  $a, b_1, b_2$ , if  $D \setminus \{a\}$  is connected,  $D \setminus \{a\}$  is a connected dominating set of  $G_a$ . Moreover, for any good quadruple  $(a', a'', b', b'')$  and a good connected dominating set  $\hat{D}$  of  $G$ , then  $\hat{D}$  is a connected dominating set of  $G_{a',a''}^{b',b''}$ .

*Proof.* We only need to show that  $D \setminus \{a\}$  is a dominating set of  $G_a$ . Note that in  $G_a$ ,  $a'$  and  $a''$  are dominated by  $b_1$  and  $b_2$  respectively, all vertices in  $N(a) \setminus \{b_1, b_2\}$  are removed, and each vertex in  $B_a$  is dominated by  $D \setminus \{a\}$  exactly as in  $G$ . Thus  $D \setminus \{a\}$  is a connected dominating set of  $G_a$ . Similarly,  $\hat{D}$  is a connected dominating set of  $G_{a',a''}^{b',b''}$ .  $\square$

**Lemma 4.** For each  $a \in A$ , if  $D'$  is a connected dominating set of  $G_a$ , then  $D' \cup \{a\}$  is a connected dominating set of  $G$ . Moreover, for each good quadruple  $(a', a'', b', b'')$ , if  $\hat{D}'$  is a connected dominating set of  $G_{a',a''}^{b',b''}$ , then  $\hat{D}'$  is a connected dominating set of  $G$ .

*Proof.* Note that in  $G_a$ , the only neighbor of  $a'$  and  $a''$  is  $b_1$  and  $b_2$  respectively, so both  $b_1$  and  $b_2$  should be in  $D'$  and  $D' \cup \{a\}$  is connected. Then in graph  $G$ , each vertex in  $N(a) \setminus \{b_1, b_2\}$  is dominated by  $a$ , and each vertex in  $B_a$  is dominated by  $D'$ , thus  $D' \cup \{a\}$  is a connected dominating set of  $G$ . Similarly,  $\hat{D}'$  is a connected dominating set of  $G$ .  $\square$

Now, we define a set  $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$ , where  $\mathcal{S}_1 = \{D_a \cup \{a\} | a \in A \text{ and } D_a \text{ is a minimum connected dominating set in } G_a\}$ , and  $\mathcal{S}_2 = \{D_{a',a''}^{b',b''} | (a', a'', b', b'') \text{ is good and } D_{a',a''}^{b',b''} \text{ is a minimum connected dominating set in } G_{a',a''}^{b',b''}\}$ .

*Remark 3.* For each  $G_a$  and  $G_{a',a''}^{b',b''}$ ,  $D_a$  and  $D_{a',a''}^{b',b''}$  may not be unique. For our purpose, however, for each  $a$ , we only need one such  $D_a$  in  $\mathcal{S}_1$ , and for each good quadruple  $(a', a'', b', b'')$ , we only need one such  $D_{a',a''}^{b',b''}$  in  $\mathcal{S}_2$ , see proof of Lemma 6 below.

**Lemma 5.**  $\mathcal{S}$  contains a minimum connected dominating set of  $G$ .

*Proof.* Let  $D$  be a minimum independent dominating set of  $G$ . We consider the following two cases.

**Case 1.** There is a circle  $a_{i_1}b_{i_1} \cdots a_{i_t}b_{i_t}a_{i_1}$  in  $D$  such that  $a_{i_1} < \cdots < a_{i_t}$ . Then for any  $a_{i_j}$ ,  $D \setminus \{a_{i_j}\}$  is still connected. Assume that  $a \in D \cap A$  and  $b_1, b_2 \in D \cap N(a)$  as in Lemma 1. For any minimum connected set  $D_a$  of  $G_a$ , by Lemma 3,  $|D_a| \leq |D| - 1$ , and by Lemma 4,  $|D| \leq |D_a| + 1$ , thus  $|D| = |D_a| + 1 = |D_a \cup \{a\}|$ . By Lemma 4 and the minimality of  $D$  in  $G$ ,  $D_a \cup \{a\}$  is a minimum connected dominating set of  $G$ , which is in  $\mathcal{S}_1$ .

**Case 2.** Otherwise, there is a good quadruple  $(a', a'', b', b'')$ . Similarly,  $D_{a',a''}^{b',b''}$  is a minimum connected dominating set of  $G$ , which is in  $\mathcal{S}_2$ .  $\square$

**Lemma 6.**  $\mathcal{S}$  is computable in  $O(|A|^2|B|^2(|A| + |B|)^4)$  time.

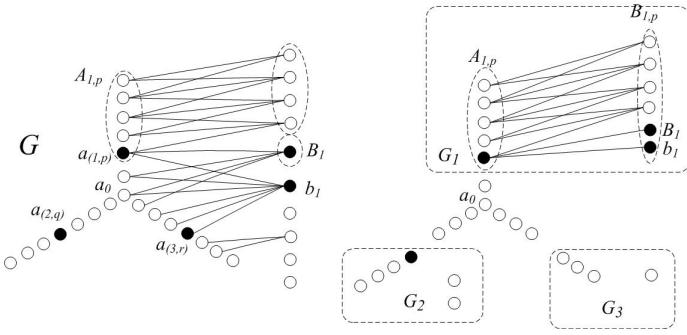
*Proof.* By Lemmm 2, for each  $a \in A$ ,  $G_a$  is convex bipartite, thus we can compute a minimum connected dominating set  $D_a$  of  $G_a$  by the known  $O((|A| + |B|)^4)$  time algorithm in [1]. As remarked in Remark 3, for each  $a$ , we only need one such  $D_a$  in  $\mathcal{S}$ . Thus, by an enumeration of all  $|A|$  vertices in  $A$ , we can compute  $\mathcal{S}_1$  in  $O(|A|(|A| + |B|)^4)$  time. Similarly, we can compute  $\mathcal{S}_2$  in  $O(|A|^2|B|^2(|A| + |B|)^4)$  time by an enumeration of all possible  $|A|^2|B|^2$  good quadruple  $(a', a'', b', b'')$ . Thus, we can compute  $\mathcal{S}$  in  $O(|A|^2|B|^2(|A| + |B|)^4)$  time.  $\square$

**Theorem 1.** *Connected domination for circular-convex bipartite graphs  $G = (A, B, E)$  with circular ordering on  $A$  is solvable in  $O(|A|^2|B|^2(|A| + |B|)^4)$  time.*

*Proof.* By Lemmas 6, we can compute the set  $\mathcal{S}$  in  $O(|A|^2|B|^2(|A| + |B|)^4)$  time. Then we pick out a minimum size set in  $\mathcal{S}$  which is a connected dominating set of  $G$ . By Lemmas 5, this set is a minimum connected dominating set of  $G$ .  $\square$

## 4 Reduction for Triad-Convex Bipartite Graphs

For a triad-convex bipartite graph  $G = (A, B, E)$ , we always assume a triad on  $A$ , so for each vertex in  $B$ , its neighborhood is a subtree in the triad. The triad is three paths with a common end, we assume that  $A = \{a_0\} \cup A_1 \cup A_2 \cup A_3$ ,



**Fig. 3.** A triad-convex bipartite graph, center  $a_0$  and  $a_{1,p}, a_{2,q}, a_{3,r}$

where for  $1 \leq i \leq 3$ ,  $A_i = \{a_{i,1}, a_{i,2}, \dots, a_{i,n_i}\}$  and  $a_0 a_{i,1} a_{i,2} \cdots a_{i,n_i}$  is a path, respectively. Note that for every  $b \in B \setminus N(a_0)$ , we have  $N(b) \subseteq A_i$  for some  $i$ .

Let  $D$  be a minimum connected dominating set of  $G$ , and  $a_{1,p}, a_{2,q}, a_{3,r}$  are the vertices nearest to  $a_0$  in  $D \cap A_i$  for  $i = 1, 2, 3$  respectively. See Figure 3.

Especially, if  $a_0 \in D$ , then we define that  $p = q = r = 0$ , and  $a_{1,0} = a_{2,0} = a_{3,0} = a_0$ . (Some of the  $p, q, r$  may not exist, but that will only simplify our tasks. The details are in below.)

**Lemma 7.**  $\{a_{1,p}, a_{2,q}, a_{3,r}\}$  dominates  $N(a_0)$  and  $\{a_{1,p}, a_{2,q}, a_{3,r}\} \subseteq N(N(a_0))$ .

*Proof.* If  $\{a_{1,p}, a_{2,q}, a_{3,r}\}$  does not dominate  $N(a_0)$ , then there exists a vertex  $x \in N(a_0)$ , such that  $\{a_{1,p}, a_{2,q}, a_{3,r}\} \cap N(x) = \emptyset$ . By property of triad-convex bipartite graphs, we then have  $D \cap N(x) = \emptyset$ , which is a paradox.

If  $\{a_{1,p}, a_{2,q}, a_{3,r}\}$  is not contained in  $N(N(a_0))$ , without loss of generality, we assume that  $a_{1,p} \notin N(N(a_0))$ . Define  $A_{1,p+} = \{a_{1,x} \in D | a_{1,x} \in A_1, x \geq p\}$ . Divide  $D$  into two parts:  $D_{1,p+} = A_{1,p+} \cup N(A_{1,p+})$  and  $D_{1,p-} = D - D_{1,p+}$ . If there exists an edge between  $D_{1,p+}$  and  $D_{1,p-}$ , there must exist a vertex  $x$  to dominate  $a_0$  and  $a_{1,p}$ , which means that  $a_{1,p} \in N(N(a_0))$ . This is a contradiction, since we assume that  $a_{1,p} \notin N(N(a_0))$ . If there is no edge between  $D_{1,i+}$  and  $D_{1,i-}$ , this also contradicts the fact that  $D$  is connected.  $\square$

Lemma 7 sets constraints on selection of  $a_{1,p}, a_{2,q}, a_{3,r}$ . Now we add vertices from  $N(a_0)$  into  $D$ . Because we need to find the connected dominating set, we must make  $a_{1,p}, a_{2,q}, a_{3,r}$  connected with help of some vertices in  $N(a_0)$ . If a vertex in  $N(a_0)$  only dominates one of  $a_{1,p}, a_{2,q}, a_{3,r}$ , then it does not make any contribution to the connection. We divide  $N(a_0)$  into the following four parts.  $B_0 = N(\{a_{1,p}, a_{2,q}, a_{3,r}\}) \setminus B_1 \setminus B_2 \setminus B_3$ .  $B_1 = N(a_{1,p}) \setminus N(\{a_{2,q}, a_{3,r}\})$ .  $B_2 = N(a_{2,q}) \setminus N(\{a_{1,p}, a_{3,r}\})$ .  $B_3 = N(a_{3,r}) \setminus N(\{a_{1,p}, a_{2,q}\})$ . If  $a_{1,p}$  does not exist, then  $B_1 = \emptyset$ . The situations for  $a_{2,q}$  and  $a_{3,r}$  are similar. By Lemma 7,  $B_0 \cup B_1 \cup B_2 \cup B_3 = N(a_0)$ . Especially, if  $a_{1,p} = a_{2,q} = a_{3,r} = a_0$ , then  $B_0 = N(a_0)$  and  $B_1 = B_2 = B_3 = \emptyset$ .

Then, we need to select vertices from  $B_0$  to make  $a_{1,p}, a_{2,q}, a_{3,r}$  connected.

**Lemma 8.** For  $D$  and  $B_0$  defined as above, we have  $|D \cap B_0| \leq 3$ .

*Proof.* We find three vertices  $b_1, b_2, b_3 \in B$ , such that  $N(b_i) \cap A_i = N(B_0) \cap A_i$  for  $i = 1, 2, 3$ . If  $C' = D \cap B_0$ ,  $|C'| \geq 4$ , we can first add  $\{b_1, b_2, b_3\}$  into  $D$ . Then if we delete the vertices in  $C'$ , then  $D$  is still connected. And all the vertices dominated by  $C'$  can be dominated by  $\{b_1, b_2, b_3\}$ . Thus we can get a new set  $D' = (D - C') \cup \{b_1, b_2, b_3\}$ .  $D'$  is also a connected dominating set, and  $|D'| < |D|$ . This contradicts the minimality of  $D$ .  $\square$

Let  $D$  be a minimum connected dominating set of  $G$ . By lemma 8, define  $C = D \cap B_0 = \{b_1, b_2, b_3\}$ ,  $N(b_i) \cap A_i = N(C) \cap A_i$  for  $i = 1, 2, 3$  respectively. Note that the situation  $b_1 = b_2$  maybe happen but it has no harm to the result.

**Lemma 9.** The induced graph of  $\{a_{1,p}, a_{2,q}, a_{3,r}, b_1, b_2, b_3\}$  is connected.

*Proof.* If not,  $D$  is not connected.  $\square$

Define  $G_1 = \{A_{1,p}, B_{1,p}, E_1\}$  where  $A_{1,p} = \{a_{1,x} \in A_{1,p} \mid x \geq p\}$ ,  $B_{1,p} = \{b \mid N(b) \subseteq A_1\} \cup B_1 \cup \{b_1\}$ , and  $E_1 = \{e \in E \mid e \text{ is incident to a vertex in } A_{1,p} \text{ and a vertex in } B_{1,p}\}$ . The definitions of  $G_2, G_3$  are similar.

**Lemma 10.**  $G_1, G_2, G_3$  are all convex bipartite.

*Proof.* Similar to the proof of Lemma 2.  $\square$

**Lemma 11.** If  $D$  is an minimum connected dominating set of  $G$  with the condition that  $a_{1,p}, a_{2,q}, a_{3,r}$  are the first vertices of each path in  $D$  and  $b_1, b_2, b_3 \in D$ , then  $D \cap (A_{1,p} \cup B_{1,p})$  is a connected dominating set of  $G_1$ . Similarly for  $G_2$  and  $G_3$ .

*Proof.* We prove by definition of connected dominating set. Because  $b_1 \in B_{1,p}$ , and  $N(b_1) \cap A_{1,p} = N(\{b_1, b_2, b_3\}) \cap A_{1,p}$ , then every vertex in  $A_{1,p}$  can be dominated by  $D \cap B_{1,p}$ . Every vertex in  $B_1 \cup \{b_1\}$  is dominated by  $a_{1,p},$ , and  $a_{1,p}$ , so every vertex in  $B_{1,p}$  can be dominated by  $D \cap A_{1,p}$ . The connection of the set is obviously. So  $D \cap (A_{1,p} \cup B_{1,p})$  is a connected dominating set. Similarly for  $G_2$  and  $G_3$ .  $\square$

**Lemma 12.** Let  $D_i$  be a dominating set of  $G_i$  with  $b_i \in D_i$  for  $i = 1, 2, 3$ , respectively. Then  $D_1 \cup D_2 \cup D_3 \cup \{a_{1,p}, a_{2,q}, a_{3,r}\}$  is a connected dominating set of  $G$ .

*Proof.* First, every vertex in  $G_1 \cup G_2 \cup G_3$  is dominated by  $D_1, D_2, D_3$ . The the rest of vertices in  $G$  can be dominated by  $\{a_{1,p}, a_{2,q}, a_{3,r}, b_1, b_2, b_3\}$ . So  $D_1 \cup D_2 \cup D_3 \cup \{a_{1,p}, a_{2,q}, a_{3,r}\}$  is a dominating set of  $G$ . Second,  $D_1, D_2, D_3$  are all connected.  $b_1 \in D_1, b_2 \in D_2, b_3 \in D_3$ , and by Lemma 9 the induced subgraph of  $\{b_1, b_2, b_3, a_{1,p}, a_{2,q}, a_{3,r}\}$  is connected. So  $D_1 \cup D_2 \cup D_3 \cup \{a_{1,p}, a_{2,q}, a_{3,r}\}$  is a connected subgraph of  $G$ .  $\square$

Define set  $S = \{D(p, q, r, b_1, b_2, b_3) \mid a_{1,p} \in A_1, a_{q,j} \in A_2, a_{3,r} \in A_3, b_1, b_2, b_3 \in N(a_0)\}$ , where  $D(p, q, r, b_1, b_2, b_3) = D_1 \cup D_2 \cup D_3 \cup \{a_{1,p}, a_{2,q}, a_{3,r}\}$ .

**Lemma 13.**  $\mathcal{S}$  contains a minimum connected dominating set of  $G$ .

*Proof.* Because of Lemma 11 and Lemma 12,  $\mathcal{S}$  contain all the minimum connected dominating set  $D$  with condition  $\{a_{1,p}, a_{2,q}, a_{3,r}, b_1, b_2, b_3\} \subseteq D$ . So the minimum connected dominating set of  $G$  must be include by  $\mathcal{S}$ .  $\square$

**Theorem 2.** *Connected Domination for triad-convex bipartite is solvable in  $O(|A|^3|B|^3(|A| + |B|)^4)$  time.*

*Proof.* Enumerating all the possible  $a_{1,i}, a_{2,j}, a_{3,k}$  cost  $O(|A|^3)$ . Enumerating suitable  $b_1, b_2, b_3$  cost  $O(|B|^3)$ . The algorithm for convex bipartite graphs runs in  $O((|A| + |B|)^4)$  time, which is the time cost of getting  $D_1, D_2$  and  $D_3$ . So our algorithm for triad-convex bipartite graph runs in  $O(|A|^3|B|^3(|A| + |B|)^4)$  time.  $\square$

**Acknowledgments.** We thank Professor Kaile Su for his encouragements and supports to this work. We also thank Professor Francis Y.L. Chin for bringing our attention to the notion of circular convex bipartite graphs during FAW-AAIM 2011.

## References

1. Damaschke, P., Müller, H., Kratsch, D.: Domination in Convex and Chordal Bipartite Graphs. *Inf. Process. Lett.* 36(5), 231–236 (1990)
2. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman and Company (1979)
3. Grover, F.: Maximum matching in a convex bipartite graph. *Nav. Res. Logist. Q.* 14, 313–316 (1967)
4. Jiang, W., Liu, T., Ren, T., Xu, K.: Two Hardness Results on Feedback Vertex Sets. In: Atallah, M., Li, X.-Y., Zhu, B. (eds.) FAW-AAIM 2011. LNCS, vol. 6681, pp. 233–243. Springer, Heidelberg (2011)
5. Jiang, W., Liu, T., Wang, C., Xu, K.: Feedback vertex sets on restricted bipartite graphs. *Theor. Comput. Sci.* (in press, 2013), doi:10.1016/j.tcs.2012.12.021
6. Jiang, W., Liu, T., Xu, K.: Tractable Feedback Vertex Sets in Restricted Bipartite Graphs. In: Wang, W., Zhu, X., Du, D.-Z. (eds.) COCOA 2011. LNCS, vol. 6831, pp. 424–434. Springer, Heidelberg (2011)
7. Liang, Y.D., Blum, N.: Circular convex bipartite graphs: Maximum matching and Hamiltonian circuits. *Inf. Process. Lett.* 56, 215–219 (1995)
8. Müller, H., Brandstät, A.: The NP-completeness of steiner tree and dominating set for chordal bipartite graphs. *Theor. Comput. Sci.* 53(2-3), 257–265 (1987)
9. Pfaff, J., Laskar, R., Hedetniemi, S.T.: NP-completeness of total and connected domination, and irredundance for bipartite graphs. Technical Report 428, Dept. Mathematical Sciences, Clemenson Univ. (1983)
10. Sampathkumar, E., Walikar, H.B.: The connected domination number of a graph. *Math. Phys. Sci.* 13(6), 607–613 (1979)
11. Song, Y., Liu, T., Xu, K.: Independent Domination on Tree Convex Bipartite Graphs. In: Snoeyink, J., Lu, P., Su, K., Wang, L. (eds.) FAW-AAIM 2012. LNCS, vol. 7285, pp. 129–138. Springer, Heidelberg (2012)
12. Wang, C., Liu, T., Jiang, W., Xu, K.: Feedback vertex sets on tree convex bipartite graphs. In: Lin, G. (ed.) COCOA 2012. LNCS, vol. 7402, pp. 95–102. Springer, Heidelberg (2012)

# On the Minimum Caterpillar Problem in Digraphs<sup>\*</sup>

Taku Okada, Akira Suzuki, Takehiro Ito, and Xiao Zhou

Graduate School of Information Sciences, Tohoku University,

Aoba-yama 6-6-05, Sendai, 980-8579, Japan

{okada,a.suzuki,takehiro,zhou}@ecei.tohoku.ac.jp

**Abstract.** Suppose that each arc in a digraph  $D = (V, A)$  has two costs of non-negative integers, called a spine cost and a leaf cost. A caterpillar is a directed tree consisting of a single directed path (of spine arcs) and leaf vertices each of which is incident to the directed path by exactly one incoming arc (leaf arc). For a given terminal set  $K \subseteq V$ , we study the problem of finding a caterpillar in  $D$  such that it contains all terminals in  $K$  and its total cost is minimized, where the cost of each arc in the caterpillar depends on whether it is used as a spine arc or a leaf arc. In this paper, we first study the complexity status of the problem with respect to the number of terminals: the problem is solvable in polynomial time for any digraph with two terminals, while it is NP-hard for three terminals. We then give a linear-time algorithm to solve the problem for digraphs with bounded treewidth, where the treewidth for a digraph  $D$  is defined as the one for the underlying graph of  $D$ . Our algorithm runs in linear time even if  $|K| = O(|V|)$ .

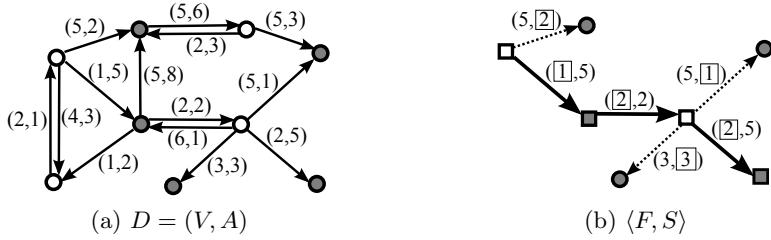
## 1 Introduction

Let  $D = (V, A)$  be a digraph whose vertex set is  $V$  and arc set is  $A$ ; we sometimes denote by  $V(D)$  the vertex set of  $D$  and by  $A(D)$  the arc set of  $D$ . A digraph  $F$  with a subset  $S \subseteq V(F)$  is called a *caterpillar*, denoted by  $\langle F, S \rangle$ , if  $S$  induces a directed path in  $F$  and every vertex in  $V(F) \setminus S$  has no outgoing arc and has exactly one incoming arc; the directed path induced by  $S$  is called the *spine* of  $\langle F, S \rangle$ . We denote by  $A_S(F, S)$  the set of all arcs on the spine of  $\langle F, S \rangle$ ; each arc in  $A_S(F, S)$  is called a *spine arc*, and each arc in  $A_L(F, S) = A(F) \setminus A_S(F, S)$  is called a *leaf arc*. Figure 1(b) illustrates a caterpillar  $\langle F, S \rangle$ , where each vertex in  $S$  is depicted by a square, each spine arc by a thick arrow, and each leaf arc by a dotted arrow.

Suppose that we are given a digraph  $D = (V, A)$  together with two cost functions  $c_S : A \rightarrow \mathbb{Z}^+$  and  $c_L : A \rightarrow \mathbb{Z}^+$ , where  $\mathbb{Z}^+$  is the set of all non-negative

---

\* This work is partially supported by JSPS Grant-in-Aid for Scientific Research, Grant Numbers 24.3660(A. Suzuki), 22700001(T. Ito) and 23500001(X. Zhou).



**Fig. 1.** (a) An instance of the minimum caterpillar problem, and (b) its optimal solution

integers. Then, for a caterpillar  $\langle F, S \rangle$  as a subgraph of  $D$ , the *cost*  $c(F, S)$  of  $\langle F, S \rangle$  is defined as follows:

$$c(F, S) = \sum_{e \in A_S(F, S)} c_S(e) + \sum_{e \in A_L(F, S)} c_L(e).$$

Let  $K \subseteq V$  be a given set of vertices, called *terminals*. Then, a caterpillar  $\langle F, S \rangle$  is called a  $K$ -caterpillar if  $K \subseteq V(F)$ . The *minimum caterpillar problem* is to find a  $K$ -caterpillar  $\langle F, S \rangle$  as a subgraph of  $D$  whose cost  $c(F, S)$  is minimized. Note that a digraph does not always have a  $K$ -caterpillar for a given set  $K \subseteq V(D)$ . In the instance of Fig. 1(a), there are five terminals, each of which is shaded, and the two costs for each arc  $e \in A$  are depicted by a pair  $(c_S(e), c_L(e))$ . Then, the  $K$ -caterpillar  $\langle F, S \rangle$  of Fig. 1(b) is an optimal solution for the instance in Fig. 1(a), whose cost is  $c(F, S) = (1 + 2 + 2) + (2 + 1 + 3) = 11$ .

The minimum caterpillar problem in digraphs is a generalization of the minimum *spanning* caterpillar problem in undirected graphs, defined as follows [3,4,9]: the *minimum spanning caterpillar problem* is the minimum caterpillar problem in which all vertices in a given digraph  $D$  are terminals, that is,  $K = V(D)$ , and there always exists an arc  $(u, v)$  if there is an arc  $(v, u)$  such that  $c_S((u, v)) = c_S((v, u))$  and  $c_L((u, v)) = c_L((v, u))$ . The minimum spanning caterpillar problem (and hence the minimum caterpillar problem) has some applications to the network design problem, the facility transportation problem, etc [4,9]. However, the minimum spanning caterpillar problem is known to be NP-hard [4], and hence the minimum caterpillar problem is also NP-hard in general. For the minimum spanning caterpillar problem on general graphs, Simonetti *et al.* [9] gave a non-polynomial-time exact algorithm, and Dinneen and Khosravani [4] studied the problem from the viewpoint of approximation. Dinneen and Khosravani [3] also gave a linear-time (exact) algorithm for (undirected) graphs with bounded treewidth.

In this paper, we give two results for the minimum caterpillar problem. We first study the complexity status of the problem with respect to the number of terminals: the problem is solvable in polynomial time for any digraph with two terminals, while it is NP-hard for digraphs with three terminals. Note that the known result of [4] does not imply the NP-hardness for a constant number of terminals. We then give a linear-time algorithm to solve the problem for digraphs with bounded treewidth. Note that, in this paper, the treewidth of a digraph

$D$  is defined simply as the one of the “underlying graph” of  $D$ , and hence it is different from [6]. (The formal definition will be given in Section 3.1.) We remark that our algorithm runs in linear time even if  $|K| = O(n)$ , where  $n$  is the number of vertices in a digraph. Therefore, our algorithm improves the known one [3] in the sense that our algorithm also solves the minimum spanning caterpillar problem in linear time for (undirected) graphs with bounded treewidth.

It is known that any optimization problem that can be expressed by Extended Monadic Second Order Logic (EMSOL) can be solved in linear time for graphs with bounded treewidth [2]. However, the algorithm obtained by this method is hard to implement, and is very slow since the hidden constant factor of the running time is a tower of exponentials of unbounded height with respect to the treewidth [7]. On the other hand, our algorithm is simple, and the hidden constant factor is just a single exponential of the treewidth.

## 2 Complexity Status

In this section, we study the complexity status of the minimum caterpillar problem with respect to the number of terminals. We omit the proofs due to the page limitation.

**Theorem 1.** *The minimum caterpillar problem is solvable in polynomial time for digraphs with two terminals.*

On the other hand, the NP-hardness can be shown by a polynomial-time reduction from the directed vertex-disjoint paths problem [5] to the minimum caterpillar problem for digraphs with three terminals.

**Theorem 2.** *The minimum caterpillar problem is NP-hard even for digraphs with three terminals.*

## 3 Algorithm for Digraphs with Bounded Treewidth

The main result of this section is the following theorem.

**Theorem 3.** *The minimum caterpillar problem can be solved in linear time for digraphs with bounded treewidth.*

In this section, we give such an algorithm as a proof of Theorem 3. Indeed, for a given digraph  $D$  and a given terminal set  $K$ , we give a linear-time algorithm which computes the minimum cost of a  $K$ -caterpillar in  $D$ ; it is easy to modify our algorithm so that it actually finds a  $K$ -caterpillar with the minimum cost.

### 3.1 Treewidth for Digraphs

We first define the notion of treewidth for an undirected graph, together with its (nice) tree-decomposition. In this paper, the treewidth for a digraph  $D$  is defined as the one for the underlying graph of  $D$ , where the *underlying graph*

$U(D)$  of a digraph  $D$  is an undirected graph whose vertex set is  $V(D)$  and edge set is  $\{\{x, y\} \mid (x, y) \in A(D) \text{ or } (y, x) \in A(D)\}$ .

Let  $G$  be an undirected graph with  $n$  vertices. We denote by  $V(G)$  and  $E(G)$  the vertex set and edge set of  $G$ , respectively. A *tree-decomposition* of  $G$  is a pair  $\langle \{X_i \mid i \in V_T\}, T \rangle$ , where  $T = (V_T, E_T)$  is a rooted tree, such that the following four conditions (1)–(4) hold [8]:

- (1) each  $X_i$  is a subset of  $V(G)$ ;
- (2)  $\bigcup_{i \in V_T} X_i = V(G)$ ;
- (3) for each edge  $\{u, v\} \in E(G)$ , there is at least one node  $i \in V_T$  such that  $u, v \in X_i$ ; and
- (4) for any three nodes  $p, q, r \in V_T$ , if node  $q$  lies on the path between  $p$  and  $r$  in  $T$ , then  $X_p \cap X_r \subseteq X_q$ .

In particular, a tree-decomposition  $\langle \{X_i \mid i \in V_T\}, T \rangle$  of  $G$  is called a *nice tree-decomposition* if the following four conditions (5)–(8) hold [1]:

- (5)  $|V_T| = O(n)$ ;
- (6) every node in  $V_T$  has at most two children in  $T$ ;
- (7) if a node  $i \in V_T$  has two children  $l$  and  $r$ , then  $X_i = X_l = X_r$ ; and
- (8) if a node  $i \in V_T$  has only one child  $j$ , then one of the following two conditions (a) and (b) holds:
  - (a)  $|X_i| = |X_j| + 1$  and  $X_i \supset X_j$  (such a node  $i$  is called an *introduce node*); and
  - (b)  $|X_i| = |X_j| - 1$  and  $X_i \subset X_j$  (such a node  $i$  is called a *forget node*.)

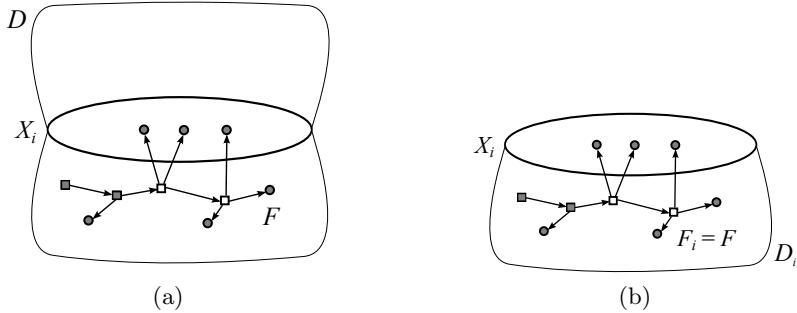
The *width* of  $\langle \{X_i \mid i \in V_T\}, T \rangle$  is defined as  $\max\{|X_i| - 1 : i \in V_T\}$ , and the *treewidth* of  $G$  is the minimum  $k$  such that  $G$  has a tree-decomposition of width  $k$ .

In this paper, we say that a digraph  $D = (V, A)$  has treewidth  $k$  if its underlying graph  $U(D)$  is of treewidth  $k$ . Since a nice tree-decomposition  $\langle \{X_i \mid i \in V_T\}, T \rangle$  of an undirected graph  $U(D)$  with bounded treewidth can be found in linear time [1], we may assume without loss of generality that a digraph  $D$  and the nice tree-decomposition  $\langle \{X_i \mid i \in V_T\}, T \rangle$  of  $U(D)$  are both given.

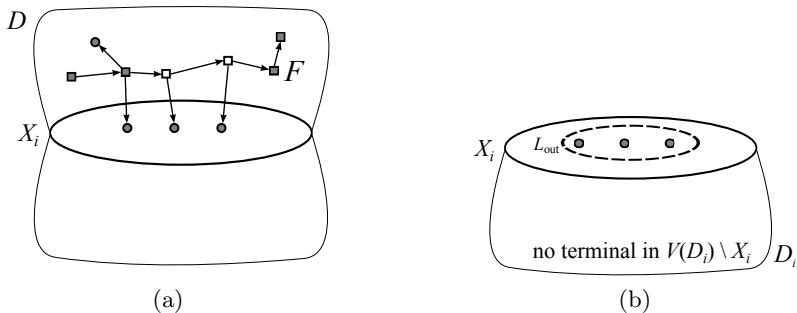
Let  $D$  be a digraph, and let  $\langle \{X_i \mid i \in V_T\}, T \rangle$  be a nice tree-decomposition of  $U(D)$ . Each node  $i \in V_T$  corresponds to a (directed) subgraph  $D_i = (V_i, A_i)$  of  $D$  which is induced by the vertices that are contained in  $X_i$  and all descendants of  $i$  in  $T$ . Therefore, if a node  $i \in V_T$  has two children  $l$  and  $r$  in  $T$ , then  $D_i$  is the union of  $D_l$  and  $D_r$  which are the subgraphs corresponding to nodes  $l$  and  $r$ , respectively. Clearly,  $D = D_0$  for the root 0 of  $T$ .

### 3.2 Main Ideas and Definitions

We first introduce some terms. Let  $\langle F, S \rangle$  be a caterpillar. Then, each vertex in  $S$  is called a *spine vertex*, while each vertex in  $V_L(F, S) = V(F) \setminus S$  is called a *leaf vertex*. Therefore, each spine arc in  $A_S(F, S)$  joins two spine vertices, and each leaf arc  $(v, w)$  in  $A_L(F, S)$  joins a spine vertex  $v \in S$  and a leaf vertex  $w \in V_L(F, S)$ ; we say that the leaf vertex  $w$  is *covered* by the spine vertex  $v$ . A spine vertex  $v \in S$  is called the *tail* of  $\langle F, S \rangle$  if the spine of  $\langle F, S \rangle$  starts from  $v$ ,



**Fig. 2.** (a) A  $K$ -caterpillar  $\langle F, S \rangle$  in  $D$  for the case where  $S \subseteq V(D_i) \setminus X_i$ , and (b) a caterpillar  $(\emptyset, S)$ -forest of  $D_i$ , where  $S = (1)$



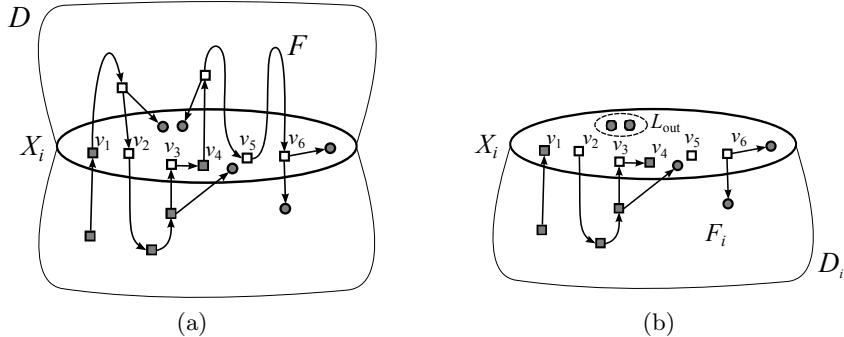
**Fig. 3.** (a) A  $K$ -caterpillar  $\langle F, S \rangle$  in  $D$  for the case where  $S \subseteq V(D) \setminus V(D_i)$ , and (b) a caterpillar  $(L_{\text{out}}, S)$ -forest of  $D_i$ , where  $L_{\text{out}} = K \cap V(D_i)$  and  $S = (0)$

while a spine vertex  $w \in S$  is called the *head* of  $\langle F, S \rangle$  if the spine of  $\langle F, S \rangle$  ends in  $w$ . The head and tail of  $\langle F, S \rangle$  are also called the *end-vertices* of  $\langle F, S \rangle$ .

We now give our main ideas. Let  $D$  be a digraph, and let  $\langle \{X_i \mid i \in V_T\}, T \rangle$  be a nice tree-decomposition of  $U(D)$ . Since we wish to find a  $K$ -caterpillar with the minimum cost, it suffices to consider  $K$ -caterpillars such that all leaf vertices are terminals in  $K$ . Consider a  $K$ -caterpillar  $\langle F, S \rangle$  as a subgraph of  $D$ , and consider the subgraph  $F_i$  of  $F$  which is induced by the vertices in  $V(F) \cap V(D_i)$  for a node  $i \in V_T$ . Then, there are the following three cases to consider, as illustrated in Figs. 2–4 where each terminal is shaded and each spine vertex is depicted by a square.

**Case (a):**  $S \subseteq V(D_i) \setminus X_i$ . (See Fig. 2.)

In this case, we claim that  $D_i$  contains the whole  $K$ -caterpillar  $\langle F, S \rangle$ , that is,  $F_i = F$ , as follows. By the definition (4) of tree-decomposition, there is no arc joining a vertex in  $V(D_i) \setminus X_i$  and a vertex in  $V(D) \setminus V(D_i)$ . Then, no spine vertex in  $S \subseteq V(D_i) \setminus X_i$  has an arc to a vertex in  $V(D) \setminus V(D_i)$ . We thus have  $V_L(F, S) \subseteq V(D_i)$ , and hence  $V(F) \subseteq V(D_i)$ . Therefore,  $D_i$  contains the whole  $K$ -caterpillar  $\langle F, S \rangle$ , as we claimed.



**Fig. 4.** (a) A  $K$ -caterpillar  $\langle F, S \rangle$  in  $D$  for the case where  $S \cap X_i \neq \emptyset$ , and (b) a caterpillar  $(L_{\text{out}}, \mathbf{S})$ -forest of  $D_i$ , where  $\mathbf{S} = (1, v_1, 0, v_2, 1, v_3, 1, v_4, 0, v_5, 0, v_6, 0)$

**Case (b):**  $S \subseteq V(D) \setminus V(D_i)$ . (See Fig. 3.)

In this case,  $D_i$  contains no spine vertex in  $S$ , but may contain leaf vertices (terminals) in  $V_L(F, S)$  which will be covered by spine vertices in  $S \subseteq V(D) \setminus V(D_i)$ . Since no spine vertex in  $S \subseteq V(D) \setminus V(D_i)$  has an arc to a vertex in  $V(D_i) \setminus X_i$ , such terminals must be in  $X_i$ . (See the three terminals surrounded by the oval  $L_{\text{out}}$  in Fig. 3(b).)

**Case (c):**  $S \cap X_i \neq \emptyset$ . (See Fig. 4.)

In this case, both  $D_i$  and  $D \setminus D_i$  may contain spine vertices, and hence  $F_i$  is not always a (single) caterpillar. However,  $F_i$  forms a *caterpillar forest*  $\langle F_i, S_i \rangle$ , where  $S_i = S \cap V(D_i)$ ; each (weakly) connected component in it is either a caterpillar or a single vertex that was a leaf vertex in  $\langle F, S \rangle$ . (Note that a single spine vertex is regarded as a caterpillar.) Consider any single leaf vertex in  $V_L(F, S) \cap V(D_i)$ . Then, similarly as in Case (b) above, it will be covered by some spine vertex in  $S \setminus V(D_i)$  and hence it must be in  $X_i$ . On the other hand, consider all caterpillars in  $\langle F_i, S_i \rangle$ . Then, we can naturally order the spine vertices in  $S_i = S \cap V(D_i)$  according to the order of the spine vertices of  $\langle F, S \rangle$ . It is easy to observe that every end-vertex of caterpillars in  $\langle F_i, S_i \rangle$  must be in  $X_i$  unless it is the end-vertex of  $\langle F, S \rangle$ . (See the end-vertices  $v_1, v_2, v_4, v_5, v_6$  in Fig. 4(b).)

Motivated by the three Cases (a)–(c) above, we classify caterpillar forests  $\langle F', S' \rangle$  in  $D_i$  into “caterpillar  $(L_{\text{out}}, \mathbf{S})$ -forests” with respect to the vertices in  $X_i$ . A terminal subset  $L_{\text{out}} \subseteq K \cap X_i$  represents the terminals that are neither spine vertices in  $S'$  nor leaf vertices covered by spine vertices in  $S' \subseteq V(D_i)$ ; and hence every vertex in  $L_{\text{out}}$  will be a leaf vertex which is covered by some spine vertex outside  $D_i$ . A “spine vector”  $\mathbf{S}$  for  $X_i$  represents the spine vertices in  $S' \cap X_i$  together with their order and connectivity: a vector  $\mathbf{S} = (a_0, v_1, a_1, v_2, a_2, \dots, v_t, a_t)$ ,  $t \geq 0$ , is called a *spine vector for*  $X_i$  if  $a_x \in \{0, 1\}$  for each index  $x$ ,  $0 \leq x \leq t$ , and  $v_x \in X_i$  for each index  $x$ ,  $1 \leq x \leq t$ . We sometimes denote by  $V(\mathbf{S})$  the set of all vertices in  $\mathbf{S}$ ; note that  $V(\mathbf{S}) = \emptyset$  if  $t = 0$ . Then, a caterpillar forest  $\langle F', S' \rangle$  as a subgraph of  $D_i$  is called a *caterpillar  $(L_{\text{out}}, \mathbf{S})$ -forest of*  $D_i$  if the following three conditions (a)–(c) hold:

- (a) if  $\mathbf{S} = (1)$ , then  $\langle F', S' \rangle$  is a  $K$ -caterpillar such that  $S' \cap X_i = \emptyset$ ;
- (b) if  $\mathbf{S} = (0)$ , then  $V(F') = L_{\text{out}}$  and  $F'$  forms an independent set; and
- (c) if  $t \geq 1$ , then the following six conditions (i)–(vi) hold:
  - (i) all terminals in  $K \cap V(D_i)$  are contained in  $V(F')$ ;
  - (ii)  $L_{\text{out}}$  forms an independent set in  $F'$ ;
  - (iii) if we remove all vertices in  $L_{\text{out}}$  from  $F'$  and add to  $F'$  a (dummy) arc from  $v_x$  to  $v_{x+1}$  for every two vertices  $v_x, v_{x+1} \in V(\mathbf{S})$  such that  $a_x = 0$ ,  $1 \leq x \leq t - 1$ , then the resulting digraph  $F''$  is a (single) caterpillar  $\langle F'', S' \rangle$ ;
  - (iv)  $S' \cap X_i = V(\mathbf{S})$ , and  $v_1, v_2, \dots, v_t$  appear on the spine of  $\langle F'', S' \rangle$  in this order;
  - (v) if  $a_0 = 0$ , then  $v_1$  is the tail of  $\langle F'', S' \rangle$ , otherwise the tail of  $\langle F'', S' \rangle$  is in  $V(D_i) \setminus X_i$ ; and
  - (vi) if  $a_t = 0$ , then  $v_t$  is the head of  $\langle F'', S' \rangle$ , otherwise the head of  $\langle F'', S' \rangle$  is in  $V(D_i) \setminus X_i$ .

For example, the caterpillar forest in Fig. 4(b) is a caterpillar  $(L_{\text{out}}, \mathbf{S})$ -forest of  $D_i$  for  $\mathbf{S} = (1, v_1, 0, v_2, 1, v_3, 1, v_4, 0, v_5, 0, v_6, 0)$ . We call the head (or the tail) of  $\langle F'', S' \rangle$  the *head* (resp., *tail*) of the caterpillar forest  $\langle F', S' \rangle$ .

Let  $\langle F', S' \rangle$  be a caterpillar  $(L_{\text{out}}, \mathbf{S})$ -forest of  $D_i$  for some pair  $(L_{\text{out}}, \mathbf{S})$ . If  $\mathbf{S} = (1)$ , then the spine vertices of  $\langle F', S' \rangle$  are in  $V(D_i) \setminus X_i$  and hence the spine of  $\langle F', S' \rangle$  cannot be extended to the outside of  $D_i$ ; we thus know that  $L_{\text{out}}$  must be the empty set and  $D_i$  must contain all terminals in  $K$ . On the other hand, if  $\mathbf{S} = (0)$ , then  $\langle F', S' \rangle$  has no spine vertex and hence we know that all terminals in  $D_i$  must be covered by spine vertices outside  $D_i$ . Therefore, we say that a pair  $(L_{\text{out}}, \mathbf{S})$  is *feasible for*  $X_i$  if it satisfies the following three conditions (a)–(c):

- (a) if  $\mathbf{S} = (1)$ , then  $L_{\text{out}} = \emptyset$  and  $K \subseteq V(D_i)$ ;
- (b) if  $\mathbf{S} = (0)$ , then  $L_{\text{out}} = K \cap V(D_i)$ ; and
- (c)  $L_{\text{out}} \cap V(\mathbf{S}) = \emptyset$ , and each vertex in  $V(\mathbf{S})$  appears exactly once in  $\mathbf{S}$ .

Then, it suffices to consider caterpillar  $(L_{\text{out}}, \mathbf{S})$ -forests of  $D_i$  only for feasible pairs  $(L_{\text{out}}, \mathbf{S})$  for  $X_i$ .

We finally define a value  $f(i; L_{\text{out}}, \mathbf{S})$  for a node  $i \in V_T$  and a pair  $(L_{\text{out}}, \mathbf{S})$ , which will be computed by our algorithm, as follows:

$$f(i; L_{\text{out}}, \mathbf{S}) = \min\{c(F', S') \mid \langle F', S' \rangle \text{ is a caterpillar } (L_{\text{out}}, \mathbf{S})\text{-forest of } D_i\},$$

where  $c(F', S')$  is the cost of a caterpillar  $(L_{\text{out}}, \mathbf{S})$ -forest  $\langle F', S' \rangle$ , that is, the total cost of all caterpillars in  $\langle F', S' \rangle$ . Let  $f(i; L_{\text{out}}, \mathbf{S}) = +\infty$  if  $D_i$  has no caterpillar  $(L_{\text{out}}, \mathbf{S})$ -forest or  $(L_{\text{out}}, \mathbf{S})$  is not feasible for  $X_i$ .

Our algorithm computes  $f(i; L_{\text{out}}, \mathbf{S})$  for each node  $i \in V_T$  and all feasible pairs  $(L_{\text{out}}, \mathbf{S})$  for  $X_i$ , from the leaves of  $T$  to the root of  $T$ , by means of dynamic programming. (However, we omit how to compute  $f(i; L_{\text{out}}, \mathbf{S})$  due to the page limitation.) Then, since  $D_0 = D$  for the root 0 of  $T$ , one can compute the minimum cost  $c(D, K)$  of a  $K$ -caterpillar in a given digraph  $D$ , as follows:

$$c(D, K) = \min f(0; \emptyset, \mathbf{S}), \quad (1)$$

where the minimum above is taken over all spine vectors  $\mathbf{S} = (a_0, v_1, a_1, \dots, v_t, a_t)$ ,  $0 \leq t \leq |X_0|$ , for  $X_0$  such that  $a_x = 1$  for all  $x$ ,  $1 \leq x \leq t - 1$ . Note that  $c(D, K) = +\infty$  if  $D$  has no  $K$ -caterpillar.

We now show that the number of all feasible pairs  $(L_{\text{out}}, \mathbf{S})$  for  $X_i$  can be bounded by a constant; this implies that our algorithm runs in linear time. Remember that  $|X_i| \leq k + 1$  for each node  $i \in V_T$ , where  $k$  is the treewidth of  $D$ . Then, there are at most  $\binom{k+1}{t} \cdot t! \cdot 2^{t+1}$  spine vectors  $\mathbf{S} = (a_0, v_1, a_1, \dots, v_t, a_t)$  for each  $t \geq 0$ . Thus, the number of all feasible pairs  $(L_{\text{out}}, \mathbf{S})$  for  $X_i$  can be bounded by

$$\sum_{t=0}^{k+1} \binom{k+1}{t} \cdot t! \cdot 2^{t+1} \cdot 2^{k+1-t} \leq (k+2)(k+1)^{k+1} \cdot 2^{k+2} = O(1).$$

## 4 Conclusion

In this paper, we first analyzed the complexity status of the minimum caterpillar problem with respect to the number of terminals. More precisely, the problem is solvable in polynomial time for any digraph with two terminals, while it is NP-hard even for digraphs with three terminals. We then gave a linear-time algorithm to solve the problem for digraphs with bounded treewidth.

## References

1. Betzler, N., Niedermeier, R., Uhlmann, J.: Tree decompositions of graphs: saving memory in dynamic programming. *Discrete Optimization* 3, 220–229 (2006)
2. Courcelle, B.: *Handbook of Theoretical Computer Science. Graph rewriting: an algebraic and logic approach*, vol. B, pp. 193–242. MIT Press (1990)
3. Dinneen, M.J., Khosravani, M.: A linear time algorithm for the minimum spanning caterpillar problem for bounded treewidth graphs. In: Patt-Shamir, B., Ekim, T. (eds.) SIROCCO 2010. LNCS, vol. 6058, pp. 237–246. Springer, Heidelberg (2010)
4. Dinneen, M.J., Khosravani, M.: Hardness of approximation and integer programming frameworks for searching for caterpillar trees. In: Proc. of CATS 2011, pp. 145–150 (2011)
5. Forutne, S., Hopcroft, J., Wyllie, J.: The directed subgraph homeomorphism problem. *Theoretical Computer Science* 10, 111–121 (1980)
6. Johnson, T., Robertson, N., Seymour, P.D., Thomas, R.: Directed tree-width. *Journal of Combinatorial Theory, Series B* 82, 138–154 (2001)
7. Lampis, M.: Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica* 64, 19–37 (2012)
8. Robertson, N., Seymour, P.D.: Graph Minors III. Planar tree-width. *Journal of Combinatorial Theory, Series B* 36, 49–63 (1984)
9. Simonetti, L., Frota, Y., de Souza, C.C.: An exact method for the minimum caterpillar spanning problem. In: Proc. of CTW 2009, pp. 48–51 (2009)

# A New Model for Product Adoption over Social Networks

Lidan Fan<sup>1</sup>, Zaixin Lu<sup>1</sup>, Weili Wu<sup>1</sup>, Yuanjun Bi<sup>1</sup>, and Ailian Wang<sup>2</sup>

<sup>1</sup> University of Texas at Dallas, Department of Computer Science,  
Richardson, Texas, USA 75080

<sup>2</sup> Taiyuan Institute of technology  
`{lidan.fan,zaixinlu,weiliwu,yuanjunbi}@utdallas.edu,`  
`{ym4008cn@yahoo.com.cn}`

**Abstract.** Building upon the observation that individuals' decisions to purchase a product are influenced by recommendations from their friends as well as their own preferences, in our work, we propose a new model that factors in people's preferences for a product and the number of his/her neighbors that have adopted this product. In our model, as in related ones, beginning with an "active" seed set (adopters), an adoption action diffuses in a cascade fashion based on a stochastic rule. We demonstrate that under this model, maximizing individuals' adoption of a product, called the product adoption maximization (PAM) problem, is NP-hard, and the objective function for product adoption is sub-modular for time T ( $T = 1, 2$ ) when the function for estimating the influence coming from neighbors is sub-linear. Hence, a natural greedy algorithm guarantees an approximation. Furthermore, we show that it is hard to approximate the PAM problem when the function for estimating the influence coming from neighbors is not sub-linear.

**Keywords:** Influence Diffusion, Product Adoption, Personal Preference, Viral Marketing, Social Networks.

## 1 Introduction

Suppose that we are trying to promote a product among a population of individuals. In order to market the product effectively with limited budget such as free samples, we strive to target a few key individuals or potential consumers who can in turn trigger out a large product adoption within their relation cycles. That is, their purchasing behaviors have an impact on their friends, who, when adopt the product, in turn affect others, and so forth, creating a cascade of decisions. This phenomenon is the "world-of-mouth" effects, coming from a multitude of relations and interactions between individuals. By virtue of these effects, as observed, people influence each other's decision to purchase a product. In this way, decisions can spread through the network from a small set of initial adopters to a potentially much larger group to an expected sense. Research in the area of viral marketing [1,2,3,4,5] takes advantage of these social network effects.

In viral marketing, one of the fundamental problems is to find a small set of users in a social network as the promotion targets, such that the adoption of a product by these users leads a large number of individuals to buy this product, that is to *maximize product adoption*. Our paper investigates this problem. Most previous papers study this as the *Influence Maximization* (IM) problem: determine a set  $S$  of  $k$  users yielding the largest expected cascade. Domingos and Richardson in [4,5] proposed and studied the IM problem under a very general descriptive model of influence propagation, merely giving heuristics for the problem. Later, Kempe completed a seminal work in [6], which obtained provable performance guarantees for approximation algorithms under the *Independent Cascade* (IC) and the *Linear Threshold* (LT) model. In subsequent works, on one hand, a large volume of algorithms [10,11,12,14] were explored to efficiently compute the expected influence for the IM problem. On the other hand, various influence diffusion models [7,8,13,16,20] were proposed to approach real-world scenarios on a network.

Among the literatures for the IM problem, two probabilistic propagation models [5,6], the IC and LT models are well studied. Both of them do not consider individuals' preferences for a product. For instance, in the IC model, a person buys a product if and only if at least one of his or her friends has bought it; and in the LT model, a person buys a product if and only if a certain number of his or her friends have bought it. However, in real world, individuals in a social network make decisions not only based on what others have done but also on their own preferences. That is, they may make decisions to purchase a product even none of his or her friends bought it. In realistic situations, two major drawbacks of the two models come out: one is that people may buy a product at any time, and the other is that people may buy a product even if no one in his or her social cycle has bought the product. To overcome the two drawbacks, we study the problem inherent in the question of how to maximize product adoption over time, considering both customers' preferences and the "world-of-mouth" effect (friends' influence), as well as approaches to seed selection.

As for the work in product adoption, in other traditional domains, Bohlen *et al.* [18] introduced five stages of product adoption. Kalish *et al.* [19] indicated that the adoption of a new product actually depends on factors such as price, individual's valuation of the product and so forth. Recently, observing these phenomena, Bhagat *et al.* [15] proposed the *Product Adoption Maximization* (PAM) problem, a variant of the influence maximization problem, in which they distinguished between influence and adoption. While in our paper, instead of focusing on analyzing differences between the two behaviors, we pay attention to factors that have impact on users decision-making: personal preference to a product and the number of friends having adopted it. Here, personal preference to a product is independent of influence from friends, making the possibility that individuals buy a product even if none of their friends buys it.

**Our Contributions.** We present a new model for the PAM problem. To the best of our knowledge, this is the first work that incorporates personal preference in product adoption procedure. In this model, when estimate a user's purchase

decision on a product, besides considering the influence coming from individuals' friends, we also include their own preferences to a product, which has not been considered in previous models. Through extensive analysis, we show that the PAM problem is NP-hard in this model. Moreover, We demonstrate that the objective function, i.e., the expected number of final product adopters (deadline  $T = 1, 2$ ), is sub-modular when the function of influence coming from friends is sub-linear. However, when the function of influence coming from friends is not sub-linear, we show that the objective function is not sub-modular any more and it is hard to find an approximation algorithm for  $T \geq 2$ .

**Organization.** The remainder of this paper is organized as follows: In section 2, we introduce the details about the new influence diffusion model. In section 3, we describe the PAM problem in our model, then we analyze the submodularity and non-submodularity of the objective function under distinct cases, and prove the NP-hardness of the PAM problem. we conclude this paper and show several directions for future work in section 4.

## 2 New Influence Diffusion Model

In this section, we introduce our new model and provide influence diffusion mechanisms under this model. Firstly, we show several models in influence diffusion and product adoption problems.

### 2.1 Several Models

The most far-reaching influence diffusion models in social networks are the IC and the LT models.

*Independent Cascade model* (IC) [5,6]. The IC model uses a sender-centered mechanism, in which each information sender independently influences its neighbors with some probability (information push style). When the influence of several senders reach a receiver at the same time, their influence is scheduled arbitrarily. The influence probability is determined by the sender (active individual) and has no relation with the receiver (inactive individual).

*Linear Threshold model* (LT) [5,6]. The LT model uses a receiver-centered mechanism, in which each information receiver adopts the information if and only if the number of its neighbors that have adopted the information exceeds certain threshold, where the threshold is treated as a random variable or fixed value (information pull style).

Being widely applied in many research works, however, both of the two models do not fully capture the dynamics of a cascade over self-interested agents, where individuals may make decisions to buy a product out of their own preference to it. As for particular models for product adoption, we introduce one among the most recent ones.

*Linear Threshold with Colors* (LT-C) [15]. This model distinguishes product adoption from influence. It assumes that in a network, there exist information

bridges, or tattlers who propagate the influence without adopting the product themselves. Furthermore, influence propagation depends on the extent to which a user likes the product. In addition to the influence weights among users, and the probability of liking a product is different for different users.

Though users in a network, under this model, is regarded as the agents having personal preferences, they are different from the ones in our model. In this model, the preferences of users for a product are still based on the experiences of their friends, in other words, their preferences are formed from the information provided by their friends, they make decisions by communicating with their friends. Nevertheless, in our model, even no friends have knowledge about a product or no one purchases this product, a user may buy it out of his/her own interest, which is completely independent of the experiences of his/her friends (know or adopt).

## 2.2 Our New Model

In the following, we provide a concrete description of our new model. Let  $G = (V, E, P(V))$  be an arbitrary undirected graph, representing an underlying social network. In this setting, the nodes in  $V$  denote a group of individuals (called agents or consumers). The edges between pairs of individuals represent the relations (friends, families, co-workers and so forth).  $P(V) = \{p(v_1), p(v_2), \dots, p(v_{|V|})\}$  is a set of personal preferences to a product. Throughout this paper, we regard “purchase”, “adopt” and “buy” as the same term, and call individuals *active* if they have the product, and *inactive* otherwise.

Assume we want to promote a new product  $P$ . Initially, there is no consumer purchases (uses) it, and individuals make their decisions to buy the product according to their own preferences. After the first round, some individuals have product  $P$ , and we say that these agents have become active, and those that do not inactive. At the second time, these active consumers will impose an impact on their friends (direct neighbors in the graph). Right now, the tendency that each individual decides to purchase the product not only has relation with their own favors, but also depends on the influence coming from their friends. This is actually the novel observation that we pick up from economic researches.

In our model, the cascade propagation unfolds in discrete time. Let  $N$  be the direct active neighbors of consumer  $v$  in the social network. At time  $t + 1$ , each inactive individual adopts a product with probability

$$In(v) + Out(v, N).$$

Here  $0 \leq In(v) + Out(v, N) \leq 1$ . The preference function  $In(v)$  for consumer  $v$  has relation with  $v$ 's age, interest, education and so forth.  $Out(v, N)$  is a function based on the  $v$ 's neighbors who have already adopted the product before  $t + 1$ . It can be a linear function, sub-linear function of  $|N|$ , such as  $\gamma(v) \times |N|^\beta$ , where  $0 < \beta \leq 1$ , and  $\gamma(v)$  is a coefficient related to  $v$ 's social relation, such as the total number of his/her friends.

Now, suppose that consumers make purchasing decisions along with time, starting from  $k$  seeds:  $S = \{s_1, s_2, \dots, s_k\}$ , then the probability that a consumer

$v \in V \setminus S$  purchases a product is  $In(v) + Out(v, Neig(v) \cap S)$ , where  $Neig(v)$  are the neighbors of  $v$ . Let  $S_t$  be a set including all the individuals that are active before  $t + 1$ . Then, at  $t + 1$ , the probability that  $v$  buys the product is  $In(v) + Out(v, Neig(v) \cap S_t)$ .

Two phenomena are ignored in previous models: First, a consumer may purchase a product out of his/her preference although no friend has this product. Second, at each time, each individual may purchase this product with a probability.

### 3 Problem Formulation and Analysis

In this section, we define the Product Adoption Maximization (PAM) problem under our new model, and demonstrate that the PAM problem is NP-hard. Furthermore, we show several results corresponding to different expressions of  $Out(v, N)$ . We find that when  $Out(v, N)$  is sub-linear, objective influence function is sub-modular for time constraint  $T = 1, 2$ , thus, a greedy algorithm is applied to guarantee an approximation factor of  $1 - 1/e$  [17] for the PAM problem. However, when  $Out(v, N)$  is not sub-linear, the objective influence function is not sub-modular any more.

**Product Adoption Maximization (PAM).** Given an undirected arbitrary graph  $G = (V, E, P(V))$ , a non-negative integer  $k$ , the problem of product adoption maximization is to select a seed set  $S$  of  $k$  vertices such that by initially activating those  $k$  individuals, the number of final users purchasing this product is maximized under our model, where  $P(V) = \{p(v_1), p(v_2), \dots, p(v_{|V|})\}$  is a set of personal preference to a product. Let  $f_T(S)$  be the number of buyers in the time duration  $T$ , then our goal is to maximize  $f_T(S)$ .

#### 3.1 Analysis

In the IC and LT models, influence process terminates at a time when there is no newly activated node. Since our model aims at maximizing the product adoption, we define a promotion duration  $T$  and only consider the influence process before deadline time  $T$ . Our first result is that

**Lemma 1.** *When  $Out(v, N)$  is sub-linear,  $f_1(S)$  is monotone increasing and sub-modular.*

*Proof.* It is easy to see that  $f_T(S)$  is monotone increasing for any  $T$ . To prove  $f_1(S)$  is sub-modular, we compare  $f_1(A) + f_1(B)$  and  $f_1(A \cup B) + f_1(A \cap B)$  for any two subsets  $A$  and  $B$  of  $V$ . Define  $p_{T,v}(S)$  to be the probability that  $v$  will adopt the product at time  $T$  when  $S$  is the seed set and  $S^*(v) = Neig(v) \cap S$ . Then  $f_1(S) = \sum_{v \in V} p_{1,v}(S)$ , in which

$$\begin{cases} p_{1,v}(S) = In(v) + Out(v, |S^*(v)|), & \text{for } v \in V \setminus S \\ p_{1,v}(S) = 1, & \text{for } v \in S \end{cases}$$

Thus, to prove  $f_1(A) + f_1(B) \geq f_1(A \cup B) + f_1(A \cap B)$ , it suffices to prove

$$p_{1,v}(A) + p_{1,v}(B) \geq p_{1,v}(A \cup B) + p_{1,v}(A \cap B), \quad (v \in V \setminus S) \quad (1)$$

Since  $(A \cup B)^*(v) = A^*(v) \cup B^*(v)$  and  $(A \cap B)^*(v) = A^*(v) \cap B^*(v)$ , and  $Out(v, |S^*(v)|)$  is a sub-linear function of  $|S^*(v)|$  for any  $v \in V$  (i.e., the highest degree of  $|S^*(v)|$  is no more than 1), it is easy to verify that

$$Out(v, |A^*(v)|) + Out(v, |B^*(v)|) \geq Out(v, |(A \cup B)^*(v)|) + Out(v, |(A \cap B)^*(v)|). \quad (2)$$

Therefore,  $f_1(S)$  is sub-modular.  $\square$

**Theorem 1.** *When  $Out(v, N)$  is sub-linear,  $f_T(S)$  is monotone increasing and sub-modular, where  $T = 1, 2$ .*

*Proof.* By Lemma 1, we know that  $f_1(S)$  is monotone increasing and sub-modular, and from its proof, we can get that, for any node  $v \in V \setminus S$ ,  $p_{1,v}(S)$  is sub-modular where  $p_{1,v}(S)$  denotes the probability that  $v$  will adopt the product at time 1 with  $S$  the seed set. Since the events that  $v$  will buy the product at time 1 and  $v$  will buy the product at time 2 are mutually exclusive, for any  $v \in V$ , we have

$$\begin{aligned} p_{2,v}(S) &= p_{1,v}(S) + (1 - p_{1,v}(S)) \cdot (In(v) + Out(v, \sum_{v' \in Neig(v)} P(v'|S, \bar{v}, 1))) \\ &= 1 - (1 - p_{1,v}(S)) \cdot (1 - (In(v) + Out(v, \sum_{v' \in Neig(v)} P(v'|S, \bar{v}, 1)))) \\ &= 1 - (1 - p_{1,v}(S)) \cdot (1 - (In(v) + Out(v, \sum_{v' \in Neig(v)} p_{1,v'}(S)))) \end{aligned} \quad (3)$$

$Out(v, \sum_{v' \in Neig(v)} P(v'|S, \bar{v}, 1))$  in the second part of Eq.3 is the function of influence coming from the neighbors who bought the product at time 1, and  $P(v'|S, \bar{v}, 1)$  denotes the event that consumer  $v'$  decides to buy a product at time 1 with  $S$  the active seed set and consumer  $v$  did not buy the product before time 1. Therefore,  $\sum_{v' \in Neig(v)} P(v'|S, \bar{v}, 1)$  is the same as  $\sum_{v' \in Neig(v)} p_{1,v'}(S)$ , and from Lemma 1, it is sub-modular for any  $v \in V$  because the class of sub-modular functions is closed under non-negative linear combination of sub-modular functions. In addition, since  $Out(v, N) = \gamma(v) \times |N|^\beta$ , where  $0 < \beta \leq 1$ ,  $Out(v, N)$  is concave for any  $N > 0$ . Thus  $Out(v, \sum_{v' \in Neig(v)} p_{1,v'}(S))$  is still sub-modular. To show  $p_{2,v}(S)$  is sub-modular, we claim that

$$\eta(S) = (1 - p_{1,v}(S)) \cdot (1 - (In(v) + Out(v, \sum_{v' \in Neig(v)} p_{1,v'}(S)))) \quad (4)$$

is super-modular. It has been shown that both  $Out(v, \sum_{v' \in Neig(v)} p_{1,v'}(S))$  and  $p_{1,v}(S)$  are monotone increasing and sub-modular, thus we have  $\eta_1(S) =$

$(1 - p_{1,v}(S))$  and  $\eta_2(S) = (1 - (In(v) + Out(v, \sum_{v' \in Neig(v)} p_{1,v'}(S))))$  are monotone decreasing and super-modular. Therefore, our claim holds because for any subsets  $A$  and  $B$  of  $V$ , we have

$$\begin{aligned}
& \eta(A \cap B) + \eta(A \cup B) - \eta(A) - \eta(B) \\
&= \eta_1(A)(\eta_2(A \cap B) - \eta_2(A)) + \eta_2(A \cap B)(\eta_1(A \cap B) - \eta_1(A)) \\
&\quad + \eta_1(A \cup B)(\eta_2(A \cup B) - \eta_2(B)) + \eta_2(B)(\eta_1(A \cup B) - \eta_1(B)) \\
&\geq \eta_1(A \cup B)(\eta_2(A \cap B) - \eta_2(A) + \eta_2(A \cup B) - \eta_2(B)) \\
&\quad + \eta_2(B)(\eta_1(A \cap B) - \eta_1(A) + \eta_1(A \cup B) - \eta_1(B)) \\
&\geq 0,
\end{aligned} \tag{5}$$

in which the first inequality holds because: first  $\eta_1(A) > \eta_1(A \cup B)$  and  $\eta_2(A \cap B) - \eta_2(A) \geq 0$ , and second  $\eta_2(A \cap B) > \eta_2(B)$  and  $\eta_1(A \cap B) - \eta_1(A) > 0$ . Therefore,  $\eta(S)$  is super-modular and  $p_{2,v}(S) = 1 - \eta(S)$  is sub-modular. By a similar argument used in the proof of Lemma 1, it can be shown that  $f_2(S)$  is monotone increasing and sub-modular.  $\square$

According to Theorem 1, with limited budget, we can apply the greedy algorithm to maximize  $f_T(S)$  ( $T = 1, 2$ ) which has a provable approximation ratio. The pseudo-code is given in Algorithm 1.

---

**Algorithm 1.** Greedy Based Seeding (GBS)

---

```

0: Input: a graph  $G$  and two parameters  $K$  and  $T$  ( $T = 1, 2$ ), in which each node
    $v$  is characterized a constant  $In(v) > 0$  and a sub-linear function  $Out(v, N)$  ( $0 \leq
   In(v) + Out(v, N) \leq 1$ ).
1: Let  $S \leftarrow \emptyset$  ( $S$  is the seed set);
2: while  $|S| < K$  do
3:   Find a node  $v$  with the maximum  $f_T(S \cup \{v\})$ ;
4: end while
4: Output:  $S$ .

```

---

**Theorem 2.** [17] If there is a value oracle for  $f$ : Given a set  $S \in V$  the oracle returns the value  $f(S)$ , the greedy gives a  $(1 - \frac{1}{e})$ -approximation for the problem of  $\max_{|S| \leq k} f_T(S)$ .

**Theorem 3.** It is NP-hard to maximize  $f_T(S)$  for  $|S| = k$  even if  $T = 1$ .

*Proof.* We prove the theorem by doing a polynomial time reduction from the 3 Dimensional Matching problem. For 3 disjoint sets  $A_1, A_2$  and  $A_3$ , let  $A$  be a subset of  $A_1 \times A_2 \times A_3$ .  $M \subseteq A$  is a 3 Dimensional Matching if any two distinct elements  $((a_{1,i}, a_{2,j}, a_{3,k}) \in M$  and  $(a_{1,l}, a_{2,m}, a_{3,n}) \in M)$  are disjoint. Given a set  $A$  and an integer  $m$ , deciding whether there exists a 3 Dimensional Matching with

$|M| \geq m$  is NP-hard. Given an instance of 3 Dimensional Matching with inputs  $A_1, A_2, A_3$  and  $A$ , we can construct a graph as follows.

1) For each triple in  $A$ , create a  $t$ -type node; and for each item in  $A_1 \cup A_2 \cup A_3$ , create a  $i$ -type node.

2) For each triple  $t = (a_1, a_2, a_3)$ , create three edges  $(t, a_1), (t, a_2)$  and  $(t, a_3)$ .

3) Let  $\Delta$  be the maximum degree in the graph. In order to make sure the adoption probability of each node is no more than 1, for each  $i$ -type node, set its own preference  $In(v) = \frac{1}{2\Delta}$  and its affected factor  $Out(v, N) = \frac{1}{2\Delta} \times N^{\frac{1}{2}}$ . For each  $t$ -type node, simply set their  $In(v) = 0$  and  $Out(v, N) = 0$  so that they will never buy the product.

4) Set the budget  $k = m$ .

It is clear that the reduction can be done in polynomial time. If the 3 Dimensional Matching is a “yes” instance, i.e., there are  $m$  disjoint elements in  $A$ . Consider the seed set which includes the  $m$   $t$ -type nodes. The expected

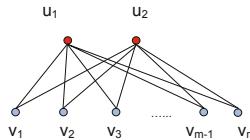
$$f_1(S) = m + 3m\left(\frac{1}{2\Delta} + \frac{1}{2\Delta} \times 1^{\frac{1}{2}}\right) = m + \frac{3m}{\Delta}. \quad (6)$$

If the 3 Dimensional Matching is a “no” instance, since  $x > x^{\frac{1}{2}}$  for any  $x \geq 1$ , it can be shown that

$$f_1(S) < m + \frac{3m}{\Delta}, \quad (\forall v \in V). \quad (7)$$

In sum, Theorem 3 is proved.  $\square$

According to Theorem 3, there is no polynomial time optimal solution for the seed selection problem even if the cycle length  $T = 1$ .



**Fig. 1.** An illustration example

**Theorem 4.** When  $Out(v, N)$  is not sub-linear,  $f_T(S)$  is not sub-modular for any  $T$ .

*Proof.* We prove Theorem 3 by giving a counterexample. As shown in Fig. 1, we construct a complete bipartite graph, in which nodes  $u_1$  and  $u_2$  belong to the upper side and other nodes belong to the lower side.

1) For nodes belong to the lower side, set  $In(v) = \alpha$  and  $Out(v, N) = \beta|S^*(v)|^\gamma$ .

2) For nodes belong to the upper side, simply set  $In(v) = 0$  and  $Out(v, N) = 0$ .

Let  $A = \{u_1\}$ ,  $B = \{u_2\}$  and  $m$  be the number of nodes in the lower side. Then  $f_T(A) = f_T(B) = 1 + m(1 - (1 - \alpha - \beta)^T)$ ,  $f_T(A \cup B) = 1 + m(1 - (1 - \alpha - \beta 2^\gamma)^T)$  and  $f_T(A \cap B) = 1 + m(1 - (1 - \alpha)^T)$ . Let  $\delta = 1 - \alpha - \beta$ , then

$$\begin{aligned} & f_T(A) + f_T(B) - (f_T(A \cup B) + f_T(A \cap B)) \\ &= 2m(1 - \delta^T) - m(1 - (\delta + \beta)^T) - m(1 - (\delta - \beta(2^\gamma - 1))^T) \\ &= m((\delta + \beta)^T + (\delta - \beta(2^\gamma - 1))^T - 2\delta^T). \end{aligned} \quad (8)$$

It is flexible to set  $\gamma$  according to  $\delta$ ,  $T$  and  $\beta$  to make Eq.8 less than zero, and when  $\gamma > \log_2(\frac{\delta - (2\delta^T + (\delta + \beta)^T)}{\beta} + 1)$ , then  $f_T(S)$  is not sub-modular. Therefore,  $f_T(S)$  is not sub-modular when  $Out(v, N)$  is an arbitrary function of  $N$ .  $\square$

According to Theorem 4, it can be shown that to approximate  $f_T(S)$  for  $|S| = k$  when  $T \geq 2$  is computational hard.

## 4 Conclusion

With respect to individual's decision to adopt a product, previous models such as the IC and LT do not consider personal preference to the product. However, our observations on real-world data show that personal preference plays a vital role in decision-making process. In this paper, we propose a novel propagation model that accounts for our observations. Under this model, we formalize the problem of Product Adoption Maximization (PAM). We demonstrate that the expected adoption spread function under this model is sub-modular within  $T$  time ( $T = 1, 2$ ) when the function of influence coming from neighbors is sub-linear over the number of active neighbors, therefore, the classic greedy algorithm is adopted. We also show that the problem is NP-hard, and when the object function of the influence imposed by neighbors is not sub-linear over the number of active neighbors, thus the objective function is not sub-modular.

Although various models have been presented for influence diffusion or product promotion, it is still a long way for us to construct realistic models. Some situations should be considered in future works: 1) personal preference may change along with time, 2) other factors exist in influence propagation among individuals. Another research direction is to validate this new model against many real data sets from diverse domains. Last but not the least, scalable algorithms are expected for our model to target the seeds efficiently in very large networks.

**Acknowledgments.** This work was supported in part by the US National Science Foundation (NSF) under Grant no. CNS-1016320 and CCF-0829993.

## References

1. Brown, J., Reinegen, P.: Social ties and word-of-mouth referral behavior. *Journal of Consumer Research* 14, 350–362 (1987)

2. Goldenberg, J., Libai, B., Muller, E.: Using complex systems analysis to advance marketing theory development: Modeling heterogeneity effects on new product growth through stochastic cellular automata. *Academy of Marketing Science Review* (2001)
3. Goldenberg, J., Libai, B., Muller, E.: Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing Letters* 12, 211–223 (2001)
4. Richardson, M., Domingos, V.: Mining knowledge-sharing sites for viral marketing. In: KDD, pp. 61–70 (2002)
5. Domingos, P., Richardson, M.: Mining the network value of customers. In: KDD, pp. 57–66 (2001)
6. Kempe, D., Kleinberg, J.M., Tardos, É.: Maximizing the spread of influence through a social network. In: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD, pp. 137–146 (2003)
7. Kempe, D., Kleinberg, J.M., Tardos, É.: Influential nodes in a diffusion model for social networks. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1127–1138. Springer, Heidelberg (2005)
8. Kimura, M., Saito, K.: Tractable models for information diffusion in social networks. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) PKDD 2006. LNCS (LNAI), vol. 4213, pp. 259–271. Springer, Heidelberg (2006)
9. Kimura, M., Saito, K., Nakano, R.: Extracting influential nodes for information diffusion on a social network. In: AAAI, pp. 1371–1376 (2007)
10. Chen, W., Wang, Y., Yang, S.: Efficient influence maximization in social networks. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD (2009)
11. Chen, W., Yuan, Y., Zhang, L.: Scalable influence maximization in social networks under the linear threshold model. In: Proceedings of the 10th IEEE International Conference on Data Mining, ICDM 2010, pp. 88–97 (2010)
12. Chen, W., Wang, C., Wang, Y.: Scalable influence maximization for prevalent viral marketing in large-scale social networks. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD, pp. 1029–1038 (2010)
13. Lu, Z., Zhang, W., Wu, W., Fu, B., Du, D.: Approximation and inapproximation for the influence maximization problem in social networks under deterministic linear threshold model. In: ICDCSW, pp. 160–165 (2011)
14. Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., VanBriesen, J., Glance, N.: Cost effective outbreak detection in networks. In: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2007), pp. 420–429 (2007)
15. Bhagat, S., Goyal, A., Lakshmanan, L.V.S.: Maximizing product adoption in social networks. In: Web Search and Data Mining, WSDM (2012)
16. Kimura, M., Saito, K., Motoda, H.: Efficient estimation of influence functions for SIS model on social networks. In: Proc. of the 21st International Joint Conference on Artificial Intelligence, pp. 2046–2051 (2009)
17. Nemhauser, G., Wolsey, L., Fisher, M.: An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming* 14, 265–294 (1978)
18. Bohlen, J.M., Beal, G.M.: The diffusion process. Spl. Report No. 18, Agri. Extn. Serv., Iowa State College (1957)
19. Kalish, S.: A new product adoption model with price, advertising, and uncertainty. *Management Science* 31(12) (1985)
20. Fan, L., Lu, Z., Wu, W., Thuraisingham, B., Ma, H., Bi, Y.: Least cost rumor blocking in social networks. In: Proceedings of the 33rd International Conference on Distributed Computing Systems, ICDCS (2013)

# Generating Uncertain Networks Based on Historical Network Snapshots

Meng Han<sup>1</sup>, Mingyuan Yan<sup>1</sup>, Jinbao Li<sup>2</sup>, Shouling Ji<sup>1</sup>, and Yingshu Li<sup>1,2</sup>

<sup>1</sup> Department of Computer Science, Georgia State University

<sup>2</sup> School of Computer Science and Technology, Heilongjiang University

{mhan7,myan2}@student.gsu.edu, jibli@hlju.edu.cn,  
sji@cs.gsu.edu, yili@gsu.edu

**Abstract.** Imprecision, incompleteness and dynamic exist in wide range of network applications. It is difficult to decide the uncertainty relationship among nodes since traditional models do not make sense on uncertain networks, and the inherent computational complexity of problems with uncertainty is always intractable. In this paper, we study how to capture the uncertainty in networks by modeling a series snapshots of networks to an uncertain graph. Since the large number of possible instantiations of an uncertain network, a novel sampling scheme is proposed which enables the development of efficient algorithm to measure in uncertain networks; considering the practical of neighborhood relationship in real networks, a framework is introduced to transform the uncertain networks into deterministic weight networks where the weights on edges can be measured as Jaccard-like index. The comprehensive experimental evaluation on real data demonstrates the effectiveness and efficiency of our algorithms.

**Keywords:** Data mining, Snapshot, Uncertain graph, Social networks.

## 1 Introduction

Networks such as the Internet, social networks, wireless networks, biological networks *etc.* are now indispensable in our daily life. Most networks are uncertain on the aspects of network settings, traffic patterns, user information etc. In the recent years, there has been tremendous interests in mining and discovering implicit knowledge from various networks.

In real life, uncertainty exists in all kinds of networks. The uncertainty may result from network components themselves or from external factors. On the one hand, most networks whose structures and features are changing all the time are dynamic. For example, in a social network, a group of colleagues form a community when they are in the same company. In due time, such a colleague relationship may be broken as some of them begin to work in another company while some of them start graduate studies. On the other hand, uncertainty is caused by the data generation process, and the variety of networks. Different data acquisition techniques and data description methods may result in incomplete

and inaccurate data which aggregates network uncertainty. Therefore, how to identify relationships in networks considering uncertainty is very stringent.

However, in practice, a clear relationship among pairs of nodes is hard to detect in huge uncertain complicated networks. Due to the increase of complexity in modern networks especially social networks (Facebook, Twitter and LinkedIn, etc.), it becomes more and more difficult to efficiently identify relationship in networks. Following are the emerging challenges.

First, even today how to model and define uncertainty in real life is still an open problem. To conduct experiments on uncertain networks, almost all the existing works are evaluated based on the PPI (Protein-protein interaction) network, which is a very famous uncertain graph database representing protein interactions for different organisms obtained from the STRING database [1]. Furthermore, although uncertainty exists, there is no representative uncertain data set can be used for applications such as social networks and wireless networks, because there is no convincing model or method to generate it. Second, uncertain graphs represented in terms of structural data is much harder to manipulate than deterministic graphs. Even if we have a reasonable uncertainty model, the cost of managing and mining such uncertainty in networks is still very expensive. To estimate an expected relationship in uncertain graph usually incurs high computation cost, and sometimes even impossible for huge networks with millions of nodes and edges. Hence, computation overhead becomes a big challenge. Third, besides uncertainty, deciding relationships among nodes itself is a challenging problem. Since different applications have various demands, the relationships among nodes are affected by many factors, which are quite hard to identify relevance among nodes. In real applications, especially in social networks, a relationship is not only reflected by the link between a pair of nodes but also effected by a node's neighbors. If two nodes have many common neighbors, there might be a more strong relationship between them even if there is no direct link exists between them. Therefore, we should take the common neighbors into consideration.

Facing the aforementioned challenges, this paper has the following contributions. First, one effective way for modeling uncertainty is to approximate the dynamic feature of a network by a static model endowed with some additional features. Therefore, we propose two basic models to describe uncertainty in dynamic networks for different applications. In these two models, historical information is utilized to predict future relationships. Second, considering the expensive cost of managing and mining uncertainty, we employ the sampling technique to take care of uncertain possible worlds. Furthermore, the Chernoff bound and the Hoeffding inequality are used to guarantee the accuracy of the obtained results. Last but not the least, we design a method for relationship detection in uncertain networks. The entities in a same community or group with relationship usually interact frequently, share similar properties and generate common features. In our solution, two-hop expectation distance are adopted to approximate the expected number of common neighbors. This method can also serve as a framework for measuring the expected number of common neighbors in

uncertain graphs. Some existing community detection, networks clustering and other algorithms designed for certain graphs can then be employed in uncertain graphs based on our framework.

The rest of this paper is organized as follows. Section 2 reviews the related works. Section 3 presents the preliminaries and problem definition. Section 4 illustrates the sampling scheme and theoretical analysis. Evaluation results based on real and synthetic data sets are shown in Section 5. Section 6 concludes our paper.

## 2 Related Work

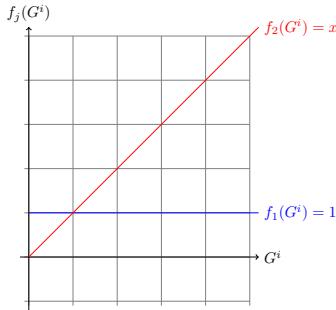
To the best of our knowledge, this work is the first one to study the uncertainty generation problem in uncertain networks. However, there do exist some works in community detection or network clustering based on relationships in traditional certain networks, and some models for uncertain data mining. Two surveys [2] and [16] present these works.

Following are the works mining and detecting relationships in certain networks, such as mining communities in YouTube [8], and mining interest groups in mobile social networks [4]. The work in [3] develops an algorithm that can identify the nodes which bridge clusters and nodes marginally connected to outliers. Since this technique needs a parameter to control the minimum similarity in a graph, the algorithms in [7] overcome the difficulty by finding only the best cluster boundaries to cluster a network. However, all the above mentioned works do not consider uncertainty in networks.

The inherent uncertainty in networks have to be considered for conducting accurate analysis and mining. The model in [6] is established for uncertain graphs (also named as probabilistic graphs) [5], in which every edge is associated with an existence probability. The following works study different issues considering uncertainty in networks. Jin *et al.* [9] introduced a sampling scheme which discovers highly reliable subgraphs with high probability. Since the shortest path in an uncertain graph is different from the ones in a deterministic graph, two new kinds of queries appear which are threshold-based shortest path queries [10] and distance-constraint teachability queries[11]. Considering the uncertainty in networks, the work in [12] introduces a framework for processing  $k$  nearest neighbor ( $k$ -NN) queries. After proposing some novel distance functions in uncertain graphs, the authors designed a sampling algorithm which can prune the search space efficiently. Unfortunately, these works and models cannot deal with community detection in uncertain networks. Moreover, all the above works do not considering the common neighbor factor, which has critical impact on identify clear relationships in uncertain graphs.

## 3 Data Model and Problem Definition

In this section, we formally present the data models considered in this paper. Similar to deterministic graphs, uncertain graphs may be undirected or directed



**Fig. 1.** Weight assignment functions for snapshot  $G^i$

and carry additional labels on edges. For simplicity and clarity, we consider undirected simple uncertain graphs. However, our discussion can be extended to directed graphs straightforwardly. We assume the edges in a graph are independent, which is common in real applications.

*Definition 1:* A dynamic network is  $\mathbb{G} = (G^0, G^1, \dots, G^t)$  where  $G^0, G^1, \dots, G^t$  are the network snapshots obtained at time  $0, 1, \dots, t$ . We use  $e^i$  ( $0 \leq i \leq t$ ) to indicate whether edge  $e$  appears in snapshot  $G^i$ .  $e^i = 1$  if  $e^i \in G^i$ , otherwise,  $e^i = 0$ .

Changes in topology mainly result in uncertainty in a network. We propose the following two different basic uncertainty models to reflect topology changes. In those two models, a weight is assigned to each snapshot  $G^i$  ( $0 \leq i \leq t$ ), and different weight assignment scenarios are applied to different models.

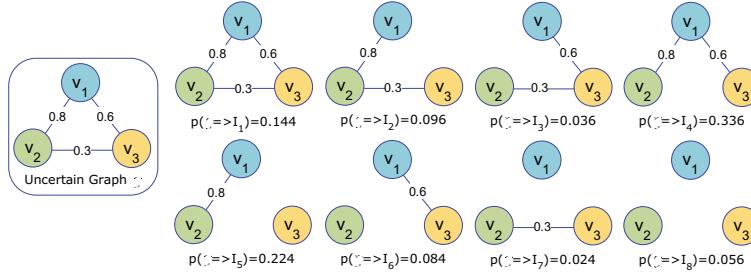
(M1) **Constant Model.** This model is used for the case where all the snapshots have the same impact on network uncertainty. Therefore, we assign the same weight to each snapshot. As shown in Fig.1,  $f_1(G^i)$  is a function which assigns constant weight 1 to each snapshot.

(M2) **Linear Model.** This model is used for the case where the snapshots have a linear changing pattern with time. Therefore, we employ a linear weight assignment scheme for this model. As shown in Fig.1,  $f_2(G^i)$  is a linear function.

For M1 or M2, the existence probability assigned to edge  $e$  is calculated as following:

$$Pr(e) = \frac{\sum_{i=0}^t e^i f_j(G^i)}{\sum_{i=0}^t f_j(G^i)} \quad (1)$$

*Definition 2:* An uncertain graph is represented by  $\mathcal{G} = (V, E, p)$ , where  $V$  is the vertices set,  $E \subseteq V \times V$  is the set of edges,  $V(\mathcal{G})$  and  $E(\mathcal{G})$  denote vertices set and edges set of  $\mathcal{G}$ ,  $p : E \rightarrow (0, 1]$  is the function assigning each edge  $e \in E$  a probability  $p(e) = Pr(e)$  obtained from Equation (1).



**Fig. 2.** Derivation of possible worlds  $I_i$  for uncertain graph  $\mathcal{G}$

Let  $I$  be a possible world instance which is a deterministic graph. As shown below,  $\mathcal{G} \Rightarrow I$  denotes that  $I$  can be generated from  $\mathcal{G}$ , and the probability of such a derivation is  $P(\mathcal{G} \Rightarrow I)$ .

$$P(\mathcal{G} \Rightarrow I) = \prod_{e \in E(I)} p(e) \prod_{e \in E(\mathcal{G}) \setminus E(I)} (1 - p(e)) \quad (2)$$

*Example 1:* Fig. 2 shows an example uncertain graph  $\mathcal{G}$ . The number marked on each edge  $e$  denotes  $p(e)$ . For  $\mathcal{G}$ , there are  $2^{|E|}$  possible worlds  $I_i (1 \leq i \leq 2^{|E|})$ . In this example, there are  $2^{|E|} = 8$  possible worlds. Each deterministic graph can be viewed as a special uncertain graph in which the existence probability of every edge is 1.

The relationship between a pair of nodes, to a considerable degree, can be determined by the number of their common neighbors. Therefore, a reasonable model for describing common neighbors in uncertain graphs is expected. Let  $N(v)$  be the neighbor set of vertex  $v$ , where neighbors mean that the existence probability between  $v$  and nodes in  $N(v)$  need to be larger than 0. As one of the most important measures in deterministic graphs, the *Jaccard index* is defined as  $\frac{N(v_i) \cap N(v_j)}{N(v_i) \cup N(v_j)}$  ranging from 0 (no overlap in the neighborhoods of  $v_i$  and  $v_j$ ) to 1 (the neighborhoods of  $v_i$  and  $v_j$  are identical). However, for uncertain graphs, it is almost impossible to derive such an index, since the relationship between every pair of nodes is imprecise.

*Definition 3:* To measure the distance between two vertices  $v_i$  and  $v_j$  in an uncertain graph, an **Expected neighbor distance**,  $Endistance(v_i, v_j)$  is defined as the expectation of the *Jaccard index*.

$$Endistance(v_i, v_j) = \text{Exp}\left(\frac{N(v_i) \cap N(v_j)}{N(v_i) \cup N(v_j)}\right) \quad (3)$$

where the function  $\text{Exp}(X)$  is the expectation of variable  $X$ .

*Definition 4:* A weight graph  $G_w = (V, E, w)$ , where  $w = Endistance(u, v)$  denotes the relationship between each pair of nodes  $u \in V(G_w)$  and  $v \in V(G_w)$ .

From the above definition, our investigated problem is defined as follows.

*Input:* A dynamic graph  $\mathbb{G} = (G^0, G^1, \dots, G^t)$ , weight assignment model type  $j$ , sampling parameters  $\epsilon$  and  $\delta$  which guarantee the accuracy.

*Output:* A weighted graph  $G_w$ .

## 4 Algorithm Framework and Theoretical Analysis

In this section, we present the framework of our algorithm. The first step is to construct an uncertain network based on dynamic snapshots; then, considering common neighbors we introduce a method to measure relationships among nodes; third, an effective sampling scheme is introduced with some optimization strategies, and a complete theoretical analysis is presented to guarantee our sampling scheme's correctness and efficiency.

### 4.1 Construct an Uncertain Network

The existing uncertain models, it is generally assumed that uncertainty exists in networks. The way they used to present uncertainty is to associate a random number between 0 to 1 to each node and/or each edge. However, except the uncertain network PPI whose probability is determined by bioexperiment, there is still no any other more reasonable model or method to present uncertainty in networks. Since the dynamic feature is one of the most important reasons resulting in uncertainty, we model uncertainty through several network snapshots coming from a dynamic network. Algorithm 1 presents the process of constructing an uncertain network.

---

#### Algorithm 1. Constructing an Uncertain Graph

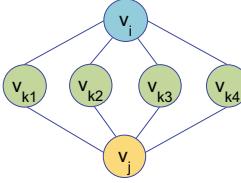
---

**input** : a set of snapshots in dynamic graph  $\mathbb{G} = (G^0, G^1, \dots, G^t)$ , the model type  $j$

**output:** uncertain graph  $\mathcal{G}$

- 1 According to model type  $j$ , use weight assignment function  $f_j(G^i)$  to assign weight to each  $(G^0, G^1, \dots, G^t) \in \mathbb{G}, \mathcal{E}_k \in E(\mathcal{G})$
  - 2 **for**  $i = 0; i \leq t - 1; i++$  **do**
  - 3   **for** each edge  $e_k$  **do**
  - 4     **if**  $e_k^i \in G^i$  **then**
  - 5        $\text{Numerator}_k += (f_j(e_k^i));$
  - 6        $\text{Denominator}_k += (f_j(e_k^i));$
  - 7    $Pr(\mathcal{E}_k) = \text{Numerator}_k / \text{Denominator}_k;$
- 

Since the required data source to construct an uncertain network is a set of snapshots of a dynamic network, the topology may change over time due to nodes' disappear or appear. We suppose that the node set includes all the



**Fig. 3.** Common neighbors in a community

nodes ever appeared in a network. If node  $i$  disappears in one snapshot, we set the weights of all the edges connected to  $i$  to 0. The computational cost of Algorithm 1 is  $O(t * n)$  where  $n$  is the number of the nodes in one snapshot and  $t$  is the number of snapshots.

#### 4.2 Measuring Relationships Among Nodes in an Uncertain Network

The number of common neighbors is one of the most important measurements for relationships among nodes. On one hand, common neighbors stand for direct relationships among nodes, since if an edge connects node  $i$  and node  $j$ , they are also common neighbors of each other (each node's neighbor set includes itself). On the other hand, the number of common neighbors also describes indirect relationships within a community. However, in an uncertain network, the concept of common neighbor is difficult to define since the direct relationship between a pair of nodes is not clear. Researchers use the expectation of an edge or path to measure a direct connection. Similarly, we use the expected number of common neighbors to represent the relationship.

In an uncertain graph  $\mathcal{G}$ , the expected number of common neighbors between node  $v_i$  and node  $v_j$  can be calculated by the expectation of the number of distinct 2-hop paths between them.

In a deterministic graph, for node  $v_i$  and node  $v_j$ , the number of common neighbors equals to the number of distinct 2-hop paths (distinct means any two paths do not have common intermediate node) between them. As shown in Fig. 3, there are four nodes ( $v_{k1}, v_{k2}, v_{k3}, v_{k4}$ ) between  $v_i$  and  $v_j$ . Obviously, there are also four distinct 2-hop paths between them correspondingly. Apparently the number of distinct 2-hop paths and the number of common neighbors is a one-one correspondence. Then we can have a deterministic graph. For  $v_i$  and  $v_j$ , a new distinct 2-hop path means adding a new node  $v_k$  as a connector between them, and  $v_k$  belongs to both  $N(v_i)$  and  $N(v_j)$ , where  $v_{k1}, \dots, v_{k4}$  are the common neighbors of  $v_i$  and  $v_j$ .

Obviously, in a deterministic graph, the number of common neighbors between two nodes corresponds to the number of 2-hop distinct paths between them. Since the expected number of common neighbors cannot be calculated directly, we use the number of 2-hop distinct paths to represent it. In an uncertain graph  $\mathcal{G}$ , a 2-hop path is a one existing in some of the possible worlds generated from  $\mathcal{G}$ .

We cannot derive whether there is a 2-hop path or not; however, we can obtain the expected existence possibility of a path according to its existence situation in each possible world.

**Lemma 1.** *In uncertain graph  $\mathcal{G}$ , the expected size of the union set of two neighbor sets belong to node  $v_i$  and node  $v_j$  can be calculated as follows.*

$$\text{Exp}(|N(v_i) \cup N(v_j)|) = \text{Exp}(|N(v_i)|) + \text{Exp}(|N(v_j)|) - \text{Exp}(|N(v_i) \cap N(v_j)|) \quad (4)$$

*Proof.* This is a simple application of the set theory.

**Theorem 1.** *In uncertain graph  $\mathcal{G}$ , the expectation of the Jaccard index  $\frac{|N(v_i) \cap N(v_j)|}{|N(v_i) \cup N(v_j)|}$  can be calculated by  $\frac{\text{ExpDCount}_{\text{Path2}}(v_i, v_j)}{\text{Exp}(|N(v_i) \cup N(v_j)|)}$ , where the numerator part  $\text{ExpDCount}_{\text{Path2}}(v_i, v_j)$  is the expected number of distinct 2-hop paths between node  $v_i$  and node  $v_j$ , and the fraction denominator part  $\text{Exp}(|N(v_i) \cup N(v_j)|)$  is the expected size of the union set of the two neighbor sets.*

*Proof.* Theorem 1 is obviously true according to and Lemma 1.

### 4.3 Sampling Possible Worlds

As mentioned, to derive  $\text{Endistance}(u, v)$  in an uncertain graph, we need to enumerate all the possible worlds to calculate the expected number of 2-hop distinct paths, then calculate the expected number of common neighbors and the expected size of the union set of the two neighbor sets.

To enumerate all the possible worlds generated from an uncertain graph  $\mathcal{G}$  is a #P-complete problem [6]. According to this fact, we cannot enumerate all the possible worlds to calculate  $\text{Endistance}(u, v)$  in an uncertain graph. We need to adopt some other more effective techniques. In this paper, we apply a sampling method to estimate  $\text{Endistance}(u, v)$ .

Now we introduce how to perform sampling of the possible worlds which follow a bernoulli distribution. Each edge in an uncertain graph either exists in a possible world with probability 1 or not shows up at all. Consider  $\mathcal{G} = (V, E, p)$  with  $n$  nodes.  $\epsilon$  and  $\delta$  are accuracy parameters where  $\epsilon$  ( $0 \leq \epsilon \leq 1$ ) and  $\delta$  ( $0 \leq \delta \leq 1$ ) denote the upper bound of relative error and failure probability respectively. The parameter  $r$  denotes the number of possible worlds. Let  $I^i$ ,  $1 \leq i \leq r$ , be a set of sampled graphs under distribution  $P$  where all  $\{I^i\}_p \in \text{Imp}(\mathcal{G})$ , where  $\text{Imp}(\mathcal{G})$  is a generated implication subspace.

The Chernoff bound gives exponentially decreasing bounds on tail distributions of sums of independent random variables [13]. We employ the Chernoff bound to reduce the number of the sampled possible worlds while guaranteeing the required accuracy.

**Lemma 2.** *Given a pair of vertices  $(u, v)$ , set  $X_i$  to be equal to 1 if there exists at least one 2-hop path from  $u$  to  $v$  in graph  $I^i$ , and 0 otherwise. According to the Chernoff bound, we get  $P(|\sum_{i=1}^r X_i - \text{Exp}(X_i)| \geq \epsilon \text{Exp}(X_i)) \leq$*

$2\exp(\frac{r \cdot \text{Exp}(X_i) \epsilon^2}{3})$ . If the number of sampled possible worlds  $r \geq \frac{3}{\epsilon^2 \text{Exp}(X_i)} \ln(\frac{2}{\delta})$ , we have

$$\left( \frac{1}{r} \left| \sum_{i=1}^r X_i - \text{Exp}(X_i) \right| \geq \epsilon \text{Exp}(X_i) \right) \leq \delta \quad (5)$$

The Hoeffding's inequality provides a method to bound the upper bound of the probability of the sum of random variables deviating from its expected value [14]. We employ the Hoeffding's inequality to further reduce the number of the sampled possible worlds while guaranteeing accuracy.

**Lemma 3.** Let  $d_i$  denote the existence probability of the 2-hop path between  $u$  and  $v$  in a possible world  $I^i$ .  $\text{Exp}(d_i)$  is the estimated expectation value according to the sampling subspace. Based on Hoeffding's inequality,  $P(\frac{1}{r} \left| \sum_{i=1}^{i=r} d_i - \text{Exp}(d_i) \right| \geq \epsilon) \leq 2\exp(\frac{2\epsilon^2}{r(n-1)^2})$ . If  $r \geq \frac{(n-1)^2}{2\epsilon^2} \ln(\frac{2}{\delta})$ , we have

$$P\left(\frac{1}{r} \left| \sum_{i=1}^{i=r} d_i - \text{Exp}(d_i) \right| \geq \epsilon\right) \leq \delta \quad (6)$$

**Theorem 2.** Consider  $p(u, v)$  as the probability distribution function respond to the distance of path between  $u$  and  $v$  in uncertain graph  $\mathcal{G}$ , set  $\widehat{p(u, v)}$  which is independent bernoulli distribution random variables as the estimator of  $p(u, v)$  come from sampling subspace in  $\{G^i\}_p$ . It is easily to prove  $\widehat{p(u, v)}$  is a unbiased estimator of  $p(u, v)$  [15]. If we sample at least  $r = \max(\frac{(n-1)^2}{2\epsilon^2}, \frac{6}{\epsilon^2} \ln(\frac{2}{\delta}))$  possible worlds, we can guarantee that:

$$P\left(\frac{1}{r} \left| \sum_{i=1}^r \widehat{p(u, v)} - p(u, v) \right| \geq \epsilon p(u, v)\right) \leq \delta \quad (7)$$

$$P\left(\frac{1}{r} \left| \sum_{\{G^i\}_p \in \text{Imp}(\mathcal{G})} \widehat{p(u, v)} - p(u, v) \right| \geq \epsilon\right) \leq \delta \quad (8)$$

*Proof.* Theorem 2 is the simple application of Lemma 2 and Lemma 3. And we change the bound from  $r \geq \frac{3}{\epsilon^2 p(u, v)} \ln(\frac{2}{\delta})$  to  $r \geq \frac{6}{\epsilon^2} \ln(\frac{2}{\delta})$  since the only attention we need to pay to is the exist probability larger than  $\frac{1}{2}$ .

We employ matrix techniques to identify 2-hop path between each pair of nodes, where a matrix is used to represent the connectivity information for a network. Consequently, we can get the 2-hop path existence by multiplying the matrix with itself, where each 1 in the result matrix  $M'$  means at least one 2-hop path exists between the corresponding pair of nodes. Based on the result matrix, our sampling algorithm can be applied to derive a weighted graph.

The computational cost of Algorithm 2 is  $O(r * n^2)$ , where  $n$  is the size of network's node set  $n = V(G_w)$  and  $r$  is the number of possible worlds which are enumerated.

**Algorithm 2.** Sampling Algorithm

---

```

input : Sampling parameters  $\epsilon$  and  $\delta$ , uncertain graph  $\mathcal{G}$ 
output: Weighted graph  $G_w$ 

1 According to  $\epsilon$  and  $\delta$ , calculate  $r$ ;
2  $i = 1$  while  $i \leq r$  do
3    $i++$ ;
4   for  $j = 0; j! = n - 1; j++$  do
5     for  $k = 0; k! = n - 1; k++$  do
6       if  $M'_{jk}! = 0$  then
7         Calculate  $Endistance(j, k) \in \mathcal{G}$  to construct  $G_w$ ;

```

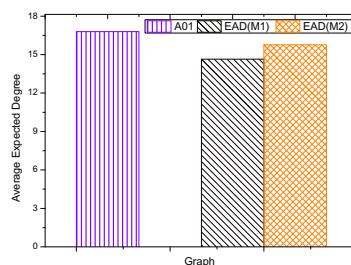
---

## 5 Experimental Evaluation

In this section, we evaluate our algorithms on the aspects of quality and efficiency. All the experiments were performed on a desktop computer with Intel(R) Core(TM)2 Quad CPU 2.83GHz and 4GB RAM. We implement all the algorithms based on BGL which is a sub library of Standard Template Library (STL)[18]. One typical dataset from SNAP (Stanford Large Network Dataset Collection)[19] was used to evaluate our algorithm. Table 1 shows the basic information of that dataset, where each row represents a network snapshot.

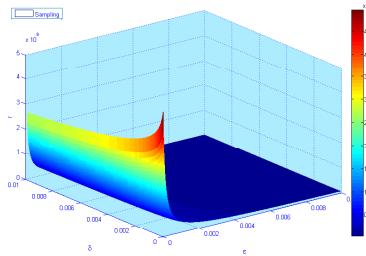
**Table 1.** Amazon Dataset

| Data             | Nodes  | Edges   | Diameter |
|------------------|--------|---------|----------|
| Amazon0302 (A02) | 262111 | 1234877 | 29       |
| Amazon0312 (A12) | 400727 | 3200440 | 18       |
| Amazon0505 (A05) | 410236 | 3356824 | 21       |
| Amazon0601 (A01) | 403394 | 3387388 | 21       |



**Fig. 4.** Effectiveness Evaluation

We first evaluate how our proposed models can depict the dynamic evolution of a network based on historical network information. We use the first three network snapshots in Table 1 are used as historical information, and employ our M1 and M2 models to generate corresponding uncertain networks. Fig. 4 shows the average degrees of the 4th network snapshot in Table 1, the expected average degree of uncertain networks generated by M1 and M2. Apparently, the generated uncertain networks by M1 and M2 are almost in accordance with the 4th network snapshot. Because the 1st input network snapshot A02 is very different from the other two, there is a little difference between the generated uncertain networks and the 4th network snapshot. M1 focuses on the overall history, while M2 considers more about the most recent situations. This is why the uncertain network generated by M2 is more proximate to the 4th network snapshot.



**Fig. 5.** Effectiveness Evaluation

Secondly, we evaluate the quality of our sampling algorithm in terms of correctness and the size of the sampling space. Base on Theorem 2,  $\epsilon$  reflects the upper bound of the difference between the sampling result and the expected result, and  $\delta$  denotes the relative error of our sampling result. Fig. 5 illustrates the different settings of  $\epsilon$  and  $\delta$ , and the resultant sampling number. In possible worlds model, the whole sampling space is  $2^{5000}$  which even cannot be accepted directly by a typical computer. However, even if the user has an extremely strict requirement of correctness, for example  $\epsilon = 0.005$  and  $\delta = 0.005$ , the sampling number is  $5.98 * 10^{12}$ . Evenly if the sampling number is only  $1.4 * 10^{10}$ , it still can be guaranteed that  $\delta$  is less than 0.08 and  $\epsilon$  is less than 0.05.

## 6 Conclusion

The importance of uncertainty in networks has been recognized in many application areas, such as social networks, wireless networks and PPI networks. In this paper, we present a framework for generating uncertain networks based on historical network snapshots. Two uncertainty construction models are presented to capture uncertainty from dynamic snapshots, and sampling techniques are also employed to improve the efficiency of the algorithm. To describe the

relationship in uncertain networks in a more practical way, 2-hop expectation distance are adopted to approximate the expected number of common neighbors. Both the theoretical analysis and our experiments demonstrate the effectiveness and efficiency of our proposed methods.

## References

1. <http://string-db.org/>
2. Lancichinetti, A., Fortunato, S.: Community detection algorithms: A comparative analysis. *Phys. Rev. E* 80, 56–117 (2009)
3. Xu, X., Yuruk, N., Feng, Z., Schweiger, T.A.J.: Scan: a structural clustering algorithm for networks. In: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 824–833 (2007)
4. Xu, K., Zhang, X.: Mining Community in Mobile Social Network. *Procedia Engineering* 29, 3080–3084 (2012)
5. Hintsanen, P., Toivonen, H.: Finding reliable subgraphs from large probabilistic graphs. *Data Min. Knowl. Disc.* 17(1), 3–23 (2008)
6. Zou, Z., Li, J., Gao, H., Zhang, S.: Mining Frequent Subgraph Patterns from Uncertain Graph Data. *IEEE Trans. on Knowl. and Data Eng.* 22(9), 1203–1218 (2010)
7. Bortner, D., Han, J.: Progressive clustering of networks using structure-connected order of traversal. In: IEEE The International Conference on Data Engineering ICDE, pp. 653–656 (2010)
8. Burton, S., Morris, R., Dimond, M., Hansen, J., Giraud-Carrier, C., West, J., Hanson, C., Barnes, M.: Public health community mining in YouTube. In: Proceedings of the 2nd ACM SIGHIT Symposium on International Health Informatics, pp. 81–90 (2012)
9. Jin, R., Liu, L., Aggarwal, C.C.: Discovering highly reliable subgraphs in uncertain graphs. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining 2011, pp. 992–1000 (2011)
10. Yuan, Y., Chen, L., Wang, G.: Efficiently answering probability threshold-based shortest path queries over uncertain graphs. In: Kitagawa, H., Ishikawa, Y., Li, Q., Watanabe, C. (eds.) DASFAA 2010. LNCS, vol. 5981, pp. 155–170. Springer, Heidelberg (2010)
11. Jin, R., Liu, L., Ding, B., Wang, H.: Distance-constraint reachability computation in uncertain graphs. *Proceedings of the VLDB Endowment* 4(9), 551–562 (2011)
12. Potamias, M., Bonchi, F., Gionis, A., Kollios, G.: K-nearest neighbors in uncertain graphs. *Proceedings of the VLDB Endowment* 3(1-2), 997–1008 (2010)
13. Chernoff, H.: A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics* 23(4), 493–507 (1952)
14. Hoeffding, W.: Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association* 58(301), 13–30 (1963)
15. Ross, S.M.: Introduction to probability models. Academic Press (2009)
16. Aggarwal, C.C., Yu, P.S.: A survey of uncertain data algorithms and applications. *IEEE Transactions on Knowledge and Data Engineering* 21(5), 609–623 (2009)
17. Strassen, V.: Gaussian elimination is not optimal. *Numer. Math.* 13(4), 354–356 (1969)
18. <http://www.boost.org>
19. <http://snap.stanford.edu/data/>

# A Short-Term Prediction Model of Topic Popularity on Microblogs

Juanjuan Zhao<sup>1</sup>, Weili Wu<sup>2</sup>, Xiaolong Zhang<sup>3</sup>, Yan Qiang<sup>1</sup>,  
Tao Liu<sup>1,\*</sup>, and Lidong Wu<sup>2</sup>

<sup>1</sup> College of Computer Science and Technology Taiyuan University of Technology, China  
`{zhaojuanjuan,qiangyan,1t0157}@tyut.edu.cn`

<sup>2</sup> Department of Computer Science, University of Texas at Dallas, Richardson, USA  
`{weiliwu,Lidong.wu}@utdallas.edu`

<sup>3</sup> College of Information Sciences and Technology Pennsylvania State University, USA  
`xiaolong.zhang@gmail.com`

**Abstract.** Online social networks can be used as networks of human sensors to detect important events. It is important to detect important events as early as possible. Microblogs provide a new communication and information sharing platform for people to report daily-life events, and express their views on various issues. Because of the quickness of microblogs, microblog data can be used to predict popular topics. In this paper, we propose a short-term prediction model of topic popularity. With data from Sina Weibo, the most popular microblog service in China, we test our algorithm and our data shows that the proposed model could give a short-term prediction on topic popularity.

**Keywords:** Social network, Microblog, Analysis of Weibo properties, Prediction model.

## 1 Introduction

Microblog[1] has become a popular social networking service in the recent years because of its ease of use and its quickness. Based on user's social relationship, microblog services offer a nearly real-time tool for people to follow the current events and comment on them. Studies have shown that people have high requirements of real-time when they browse social network websites. They are more inclined to concern with what is happening. As a result, it will be very useful to speed up the analysis of Weibo processing speed. Meanwhile, it may also bring about commercial impact on microblogging itself. Currently, Twitter's active user number per month has reached 200 million. In China, Sina Weibo, the most influential social network service, enjoys more than 300 million users.

Identifying potentially hot topics in microblogs is important. Knowing hot topics, users can better monitor microblogging opinions. Also, the information of hot topics

---

\* Corresponding author.

can provide guidance for the site maintenance, such as better coping with unexpected events.

In this paper, we propose a short-term prediction model of topic popularity based on Sina Weibo data. We first build a topic spreading model, and analyze which Weibo data features can have influence on the spreading of topics. Then, we develop the model of short-term prediction on topic popularity. Under this model, we examine the influences of individual Weibo data feature and test the performance of our prediction model on a real Sina Weibo dataset.

The rest of this paper is structured as follows: Section 2 presents the related work. Section 3 presents our model of short-term prediction on topic popularity. Section 4 shows the results obtained and finally, Section 5 exposes our conclusions and future work.

## 2 Previous works

In recent years, much research has been done on microblog topic prediction, mostly based on the research on hot topic finding. Chen et al. [2] used an “Aging Theory” to build a model to find and track topics by analyzing the temporal characteristics of topics. Some research relied on data features, such as user’s interest degree on the topic [3], the number of comments, and the number of participants [4-6], to analyze the spreading of topic. Jamali et al. [7] also used the comment data and social network derived features to predict the popularity of online contents with a classification and regression framework. Gomez et al. [8] proposed a model to predict short-term and long-term user behaviors by analyzing the temporal features of user comment activities on Slashdot. Cheng et al. [9] used the BPNN model to predict changes of network topics. Liu et al. [10] used data-mining techniques to extract keyword-based user interests from microblog contents. Nikolov and Shah [13] developed a new algorithm that can predict which Twitter topics will trend, with an accuracy as high as 95% and a speed as fast as in average one and a half hours earlier than Twitter’s own algorithm.

Although research above has made some achievements in the field of topics prediction, there are still many unsolved problems. One of these problems is that existing research has a scale limitation of their datasets. When the dataset is big enough, noisy data would be significant. Removing such noisy data requires heavy computation costs. Another problem is how to keep a high accuracy of topic prediction. Meanwhile, most research is on Twitter data, and little research has been done on Sina Weibo data, which has shown some different features from that in Twitter [15].

## 3 Method

In this section, we first introduce a topic spreading model, and then present the analysis of data features of Sina Weibo. Finally, we propose our model of short-term prediction on topic popularity.

### 3.1 Analysis of Topic Spreading Model

In this paper, we build a social network by setting a user in Weibo as a node and a user relationship as an edge. Weibo messages are propagated along the edges. Zhang et al. [15] argued that nodes in a microblog social network can be divided into three categories: communication nodes, uninfected nodes and immune nodes.

The propagation rules are: If a communication node contacts an uninfected node, uninfected node becomes a retweet node with a probability  $p_1$ . If a communication node contacts an immune node, then the communication node become an immune node with a probability  $p_2$ . A communication node does not spread endlessly; it would stop spreading with a certain speed  $v$ , and becomes an immune node without contacting other nodes.

Assume a node  $j$  is not infected at time  $t$ ,  $p_{ii}^j$  refers to the probability of node  $j$  being uninfected during time  $[t, t + \Delta t]$ , and  $p_{is}^j$  refers to the probability of node  $j$  becoming a communication node during time  $[t, t + \Delta t]$ .

$$p_{ii}^j = (1 - \Delta t p_1)^g \quad (1)$$

$g=g(t)$  represents the number of communication nodes in the neighborhood of node  $j$  at the time  $t$ .

Suppose node  $j$  has  $k$  edges,  $g$  is a random variable with binomial distribution as follows:

$$\Pi(g, t) = \binom{k}{g} \omega(k, t)^g (1 - \omega(k, t))^{k-g} \quad (2)$$

Where  $\omega(k, t)$  refers to the probability of an uninfected node with  $k$  edges connecting a communication node at time  $t$ .

$$\omega(k, t) \approx \sum_{k'} p(k'|k) \rho^s(k', t) \quad (3)$$

Where the correlation function with the  $p(k'|k)$  degree is the probability of the node with  $k$  degree being adjacent to a node with the  $k'$  degree;  $\rho^s(k', t)$  is the density of communication nodes with the  $k'$  degrees at time  $t$ . Thereby obtained  $\bar{p}_{ii}(k, t)$ , the average transition probability of the node with  $k$  degree being uninfected during time  $[t, t + \Delta t]$ , as follows:

$$\bar{p}_{ii}(k, t) = (1 - p_1 \Delta t \sum_{k'} p(k'|k) \rho^s(k', t))^k \quad (4)$$

$\bar{p}_{ss}(k, t)$ , the average transition probability of the node with  $k$  degree being infected during time  $[t, t + \Delta t]$  is as follows:

$$\bar{p}_{ss}(k, t) = (1 - p_2 \Delta t \sum_{k'} p(k'|k) \rho^r(k', t))^k \times (1 - v \Delta t) \quad (5)$$

The transition probability of a communication node becoming immune node is  $\bar{p}_{sr}(k, t) = 1 - \bar{p}_{ss}(k, t)$ .

Assuming  $N(k, t)$  is the total number of nodes with  $k$  degree in network at time  $t$ ,  $I(k, t)$ ,  $S(k, t)$ ,  $R(k, t)$  respectively refers to the number of uninfected nodes, communication nodes and immune nodes with  $k$  degree in network at time  $t$ ,

$$I(k, t) + S(k, t) + R(k, t) = N(k, t) \quad (6)$$

The change of the number of uninfected nodes with  $k$  degree in network during time  $[t, t + \Delta t]$  is as follows:

$$I(k, t + \Delta t) = I(k, t) - I(k, t)(1 - \bar{p}_{ii}(k, t)) \quad (7)$$

Similarly, the change of the number of communication nodes and immune nodes with  $k$  degree in network during time  $[t, t + \Delta t]$  are as follows:

$$S(k, t + \Delta t) = S(k, t) + I(k, t)(1 - \bar{p}_{ii}(k, t)) - S(k, t)(1 - \bar{p}_{ss}(k, t)) \quad (8)$$

$$R(k, t + \Delta t) = R(k, t) + S(k, t)(1 - \bar{p}_{ss}(k, t)) \quad (9)$$

From  $I(k, t)$  and  $I(k, t + \Delta t)$ , we can obtain:

$$(I(k, t + \Delta t) - I(k, t))/N(k, t)\Delta t = I(k, t)/N(k, t) \cdot \Delta t[1 - \bar{p}_{ii}(k, t)] \quad (10)$$

When  $\Delta t \rightarrow 0$ , using Taylor expansion on the right of equation (10), we then can obtain:

$$d\rho^i(k, t)/dt = -kp_1\rho^i(k, t) \sum_{k'} p(k'|k)\rho^s(k', t) \quad (11)$$

Similarly,

$$\begin{aligned} d\rho^s(k, t)/dt &= \\ kp_1\rho^i(k, t) \sum_{k'} p(k'|k)\rho^s(k', t) - kp_2\rho^s(k, t) \sum_{k'} p(k'|k)\rho^r(k', t) - v\rho^s(k, t) \end{aligned} \quad (12)$$

$$d\rho^r(k, t)/dt = -kp_2\rho^i(k, t) \sum_{k'} p(k'|k)\rho^r(k', t) + v\rho^s(k, t) \quad (13)$$

The simultaneous equations set obtained by equation (11), (12), (13) refer to the relationship when the density of uninfected nodes, communication nodes and immune nodes varying with time.

### 3.2 Analysis Method of Weibo Properties

Sina Weibo is different from Twitter. Fan et al. [15] investigated topological characteristics and user behavior patterns in Sina Weibo, more details can be found in their paper. To analyze the influence of each topic's attributes, we introduce a main component analysis method: combining some attributes as a complex one. The combination attribute is called the main component.

The specific step of the proposed method is as the following:

Firstly, we standardize data collection of original attribute value,  $n$  samples ( $x_i = (x_{i1}, x_{i2}, \dots, x_{ip})^T, i = 1, 2, \dots, n$ ) from  $p$ -dimensional random vector

$(x = (x_1, x_2, \dots, x_p)^T)$ . Then we construct sample matrix and standardized transform sample matrix elements as follows:

$$\bar{x}_{ij} = (x_{ij} - \bar{x}_j) / S_j, i = 1, 2, \dots, n, j = 1, 2, \dots, p \quad (14)$$

Value  $\bar{x}_{ij}$  is observational data after center standardization. Value  $\bar{x}_j$  is the sample mean of variable  $x_j$ . Value  $S_j$  is the Sample standard deviation of variable  $x_j$ . Transformed  $\bar{x}_{ij}$  become matrix  $\tilde{X} = [\bar{x}_{ij}]_{n \times p}$ .

Secondly, according to formula  $R = \frac{1}{n} \tilde{X}^T \tilde{X}$ , we compute correlation matrix of standardized matrix  $\tilde{X}$ . Elements in matrix  $R = [r_{ij}]_{m \times m}$  are:

$$r_{ij} = (\sum_{k=1}^n (x_{ki} - \bar{x}_i)(x_{kj} - \bar{x}_j)) / \sqrt{\sum_{k=1}^n (x_{ki} - \bar{x}_i)^2 \sum_{k=1}^n (x_{kj} - \bar{x}_j)^2} \quad (15)$$

When  $r_{ij} = 1$ , relationship between variable  $x_i$  and variable  $x_j$  is positive linear correlation; when  $r_{ij} = 0$ , they are irrelevant; and when  $r_{ij} = -1$ , they are negative linear correlation.

Thirdly, we compute the characteristic equation of sample correlation matrix R,  $|R - \lambda I_p| = 0$ . We can get characteristic root, and then determine the main component. According to inequality  $\sum_{j=1}^m \lambda_j / \sum_{j=1}^p \lambda_j \geq 0.60$ , value m can be determined, and the utilization of information is more than 60%. For each  $\lambda_j, j = 1, 2, \dots, m$ , we can get unit eigenvectors  $b_j$  from equation  $Rb = \lambda_j b$ .

Then, we transform the standardized indicator variables into main component:  $U_{ij} = \bar{x}_i b_j, j = 1, 2, \dots, m$ .

Finally, we evaluate each main component comprehensively and compute weighted sum. Then the final evaluation value can be computed. Weighted value equals the contribution rate of each main component.

After these step described above, we can analyze which properties will have an impact on spreading of the topics. We introduce “User influence function” (*user*) , which equals value of step (5), to describe user influence.

### 3.3 Model of Short-Term Prediction on Topic Popularity

Weibo topics are triggered by a number of reasons and conditions. It can happen at any specific time and place, and will cause some other results. On the one hand, Weibo topics have strong timelines and lifespan [2]. On the other hand, besides the time feature, there are some other features, for instance, space feature which reflects the spread process of Weibo topics on the Internet. All of these will draw the spreading route of the new topic.

The spreading of each topic has a “growth factor”. The spreading speed of topics will grow along with the growth factor’s increase. There is a maximum spreading speed of each topic. The spreading speed will slow down when it reaches its limit value. The spreading of each topic has an “attenuation factor”. People’s interest on a topic will gradually decay and shift to other topics.

According to the analysis above, we build a model to describe the spreading speed of Weibo topic in pace of time. Set  $v(t)$  represents topic's spreading speed in time  $t$ ;  $P(t)$  represents topic's growth factor in time  $t$ ; and  $M$  the maximum spreading speed, i.e. the upper limit. Parameter  $\lambda$  is the “attenuation factor” of topic,  $\lambda > 0$ . Function  $r(v)$  is the net growth rate of topic's spreading speed.  $UIF(user)$  is “User influence function” that describes in Section 3.2. This function indicates the influence of growth factor  $P(t)$  to spreading speed  $v(t)$ .

$$r(v) = UIF(user) \cdot (1 - v(t)/M) \quad (16)$$

Then, we can build a differential equation model:

$$dv(t)/dt = r(v) \cdot P(t) - \lambda \cdot v(t) \quad (17)$$

Among the equation,  $v(t) = dN(t)/dt$ ,  $N(t)$  is the sum of the browse number and reply number up to time  $t$ .  $v(t)$  means the change of  $N(t)$  from  $t - 1$  to  $t$ .

When the growth factor  $P(t)$  equals 0 and spreading speed reaches its upper limit value  $M$  ( $v(t) = M$ ), topic's spreading speed is a decreasing function along with time. From the formula above, we can see that:

$$dv(t)/dt = -\lambda \cdot v(t) \quad (18)$$

When the growth factor  $P(t)$  does not equal 0 and belongs to constant, set  $a = R \cdot P(t)/M + \lambda$ ,  $b = R \cdot P(t)$ , we can put these parameter into the differential equation model and get the equation of spreading speed  $v(t)$ :

$$v(t) = b/a \cdot (1 - e^{-at}) + v(0) \cdot e^{-at} \quad (19)$$

When the growth factor  $P(t)$  belongs to variable,  $P(t)$  will increase first and then reduce. The form of Function  $P(t)$  is :

$$\begin{cases} \frac{dP(t)}{dt} = -2(t - t_m), & P(t) > 0 \\ P(t) = 0, & \text{otherwise} \end{cases} \quad (20)$$

In this formula, value  $t_m$  represents the time when  $P(t)$  reaches its maximum.

## 4 Experiments

### 4.1 Dataset

To study and analyze Sina microblogs, we use a microblog dataset from Sina Weibo. We focus on the analysis of the user attributes and examine the performance of topic popularity prediction. This dataset has 434,512 microblogs sent by 234,512 users

from May. 2010 to Jan. 2011. It includes two types of data: 1) user basic properties, such as ID, Name, Gender, user verification flag, fans, followings, and topics; 2) topic content properties, such as number of “@”, the number of retweets, the number of comments, image or video, as shown in Table 1:

**Table 1.** Relevant properties of Weibo topic

| Property   | Description                   |
|------------|-------------------------------|
| #Fan       | Number of fan                 |
| #Following | Number of following           |
| #Topic     | Number of topic               |
| #@         | Number of mentioned user      |
| VFlag      | Whether is authenticated user |
| Gender     | Gender                        |
| Image      | Whether include images        |
| Video      | Whether include videos        |
| Time       | Tweet or retweet time         |

#### 4.2 Influence Factors to Topics Spreading

According to the method and computing step of the main component, eigenvalues and the contribution rate of each main component of the topic popularity can be calculated as shown in Table 2. From the table, we can find out that the sum of contribution rate of the first four main components has reached 76.26%. We only need to compute the first four main components, represented as  $m_1$ ,  $m_2$ ,  $m_3$  and  $m_4$ . Then, we calculate corresponding eigenvectors according to each eigenvalue, and compute the load which influences factors to each main component. We can get the load matrix of main component, as is shown in Table 3.

**Table 2.** Contribution rate of each factor

| Factor | Eigenvalue | Variance (%) | Accumulation variance (%) |
|--------|------------|--------------|---------------------------|
| 1      | 3.98       | 24.60        | 24.60                     |
| 2      | 3.17       | 19.59        | 44.19                     |
| 3      | 2.89       | 17.86        | 62.05                     |
| 4      | 2.30       | 14.21        | 76.26                     |
| 5      | 1.23       | 7.61         | 83.87                     |
| 6      | 1.02       | 6.30         | 90.17                     |
| 7      | 0.82       | 5.07         | 95.24                     |
| 8      | 0.77       | 4.76         | 100.00                    |

Table 3 describes the load size of influencing factors to the first four main components. The first Main component focuses on user properties and celebrity, such as “#Fan”, “#Following” and “#Topic”. These properties have large load size. Their values are “0.60”, “0.28”, and “0.23”. The result shows that the topic popularity is

associated with the tweets by celebrities. The second one focuses on user properties and ordinary user. The third one focuses the text content of topics, for instance, whether there is Image or video. Their values are “0.50”, and “0.41”. The last main component also focuses on topic, but with an emphasis on the number of mentioned people. Its load is “0.50”. To sum up, all of these situations imply that the number of fans, whether there is Image or video and the number of mentioned people may play an important role in spreading a topic.

**Table 3.** Load of influencing factors to each main component

| Property   | Factor 1 | Factor 2 | Factor 3 | Factor 4 |
|------------|----------|----------|----------|----------|
| #Fan       | 0.60     | 0.48     | 0.53     | 0.50     |
| #Following | 0.28     | 0.35     | 0.11     | 0.20     |
| #Topic     | 0.23     | 0.13     | 0.26     | 0.23     |
| #@         | 0.35     | 0.30     | 0.32     | 0.50     |
| VFlag      | 0.70     | 0.11     | 0.18     | 0.50     |
| Gender     | 0.05     | 0.10     | -0.03    | -0.03    |
| Image      | -0.12    | 0.31     | 0.50     | 0.28     |
| Video      | -0.14    | 0.27     | 0.41     | -0.31    |

From the analysis above, we can conclude that the number of fans, the number of followings, the number of mentioned people, whether is authenticated user and whether there is Image or video play an important role in spreading a topic. Now, we can set the “User Influence function” which is mentioned in Section 3.2 as the following:

$$UIF(user) = \theta_1 \times (\#Fan) + \theta_2 \times (\#Following) + \theta_3 \times (\#@) + \theta_4 \times (VFlag) + \theta_5 \times (Image) + \mu \quad (21)$$

The values of *VFlag* and *Image* can be set to 0 or 1.

Using real dataset as training samples and a multiple linear regression method, we compute the values of coefficient  $\theta_i$ :  $\theta_1 = 0.68$ ,  $\theta_2 = 0.13$ ,  $\theta_3 = 3.21$ ,  $\theta_4 = 15.03$ ,  $\theta_5 = 7.54$ . Putting these coefficients into the formula above, we can get the user influence value.

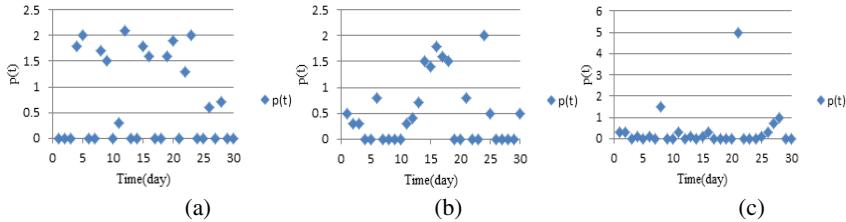
### 4.3 Short-Term Prediction on Topic Popularity

To exam the performance of the proposed popularity prediction model based on user properties, we select three representative topics. The time span is one month.

In Topic 1, we chose an entertainment star “Yao Chen” as the topic. Yao Chen has the most followers in Sina Weibo. The feature of this topic changes every day with a high growth rate and high decay rate. The initial conditions are  $\lambda = 0.7$ ,  $M = 8000$ . Figure 1 (a) shows the changes of growth factor  $P(t)$  of this topic in one month.

For Topic 2, we chose “Spring Festival travel season”. This topic becomes very hot at a specific time period. It will change relatively smooth. The initial conditions are  $\lambda = 0.5$ ,  $M = 15000$ . Figure 1 (b) shows the change of growth factor  $P(t)$  of topic 2 in one month.

For Topic 3, we chose “earthquake”. This topic belongs to unexpected events. It will suddenly appear and disappear. The initial conditions are  $\lambda = 0.9$ ,  $M = 35000$ . Figure 1 (c) shows the changes of growth factor  $P(t)$  of topic 3 in one month.



**Fig. 1.** Curve of growth factor  $P(t)$  of topic 1 (a), topic 2 (b), and topic3 (c)

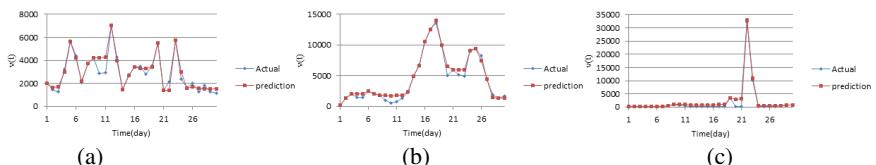
The steps of predicted topic’s spreading speed are: Compute the spreading speed  $v_A(t)$  of topic by actual data; Compute the growth factor  $P(t)$  by the spreading speed  $v_A(t)$ ;

$$P(t) = (dv_A(t)/dt + \lambda \cdot v_A(t))/(UIF(user) \cdot (1 - v_A(t)/M)) \quad (22)$$

Put this  $P(t)$  into the differential equation model mentioned in Section 4.3, formula (17). Then, we can get the prediction topic spreading speed  $v_p(t)$ .

$v_p(t)$  describes the development process of topic’s spreading speed. We can use the proposed model to give short-term forecasts on the topic of the spreading speed.

Here, we compare between the actual speed curve and the prediction speed curve of each topic in Figure 2 (a), (b) and (c). From the figures we can conclude that the prediction speed curve nicely reflects the real speed. We can use the prediction curve to give short-term predictions of topic popularity.



**Fig. 2.** Comparison between actual speed curve and prediction speed curve of Topic 1 (a), Topic 2 (b), Topic 3 (c),

## 5 Conclusions and Future Work

In this paper, we presented and evaluated a short-term prediction model of topic popularity for Sina Weibo. We introduced a topic spreading model and analyzed which Weibo properties have influence to the spreading of topics. Additionally, a model of short-term prediction on topic popularity was proposed. After testing our method on real Sina Weibo dataset, we concluded that this method could give a short-term prediction on topic popularity.

There are still some limitations in this proposed model. For example, when facing some frequently small ups and downs, the result of this model is not very accurate. Our future work is to further improve this method by increasing its prediction accuracy on these types of contents.

**Acknowledgements.** This study was supported by the National Natural Science Foundation of China (Grant No. 61202163, 61240035); Natural Science Foundation of Shanxi Province (Grant No. 2012011015-1) and Programs for Science and Technology Development of Shanxi Province (Grant No. 20120313032-3). This work was also supported in part by the US National Science Foundation (NSF) under Grant no. CNS-1016320 and CCF-0829993.

## References

1. Zhang, C., Sun, J.: Large Scale Microblog Mining Using Distributed MB-LDA. In: WWW 2012 Companion, pp. 1035–1042 (2002)
2. Chen, C.C., Chen, Y.T., Chen, M.C.: An aging theory for event life-cycle modeling. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 37(2), 237–248 (2007)
3. Pon, R.K., Cardenas, A.F., Buttler, D.J., Critchlow, T.J.: Measuring the interestingness of articles in a limited user environment. Information Processing & Management 47(t), 97–116 (2011)
4. Lu, D., Li, Q.: Exploiting semantic hierarchies for flickr group. In: An, A., Lingras, P., Petty, S., Huang, R. (eds.) AMT 2010. LNCS, vol. 6335, pp. 74–85. Springer, Heidelberg (2010)
5. Negoescu, R.A., Gatica-Perez, D.: Modeling flickr communities through probabilistic topic-based analysis. IEEE Transactions on Multimedia 12(5), 399–416 (2010)
6. Zhou, Y., Guan, X., Zheng, Q., Sun, Q., Zhao, J.: Group dynamics in discussing incidental topics over online social networks. IEEE Network 24(6), 42–47 (2010)
7. Jamali, S., Rangwala, H.: Digging Digg: Comment mining, popularity prediction, and social network analysis. In: International Conference on Web Information Systems and Mining, Shanghai, pp. 32–38 (2009)
8. Gomez, V., Kaltenbrunner, A., Lopez, V.: Statistical analysis of the social network and discussion threads in Slashdot. In: Proceedings of the 17th International Conference on World Wide Web, Beijing, China, pp. 645–654 (2008)
9. Cheng, J.J., Liu, Y., Cheng, H., Zhang, Y.C., Si, X.M., Zhang, C.L.: Growth trends prediction of online forum topics based on artificial neural networks. Journal of Convergence Information Technology 6(10), 87–95 (2011)

10. Liu, Z., Chen, X., Sun, M.: Mining the interests of Chinese microbloggers via keyword extraction. *Frontiers of Computer Science in China* 6(1), 76–87 (2012)
11. Kooti, F., Mason, W.A., Gummadi, K.P., Cha, M.: Predicting emerging social conventions in online social networks. In: CIKM, pp. 445–454 (2012)
12. Pennacchiotti, M., Popescu, A.-M.: Democrats, Republicans and Starbucks Afficionados: User Classificationin Twitter. In: Proc. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (2011)
13. Nikolov, S., Shahy, D.: A Nonparametric Method for Early Detection of Trending Topics. In: WIDS (2012)
14. Mathioudakis, M., Koudas, N.T.: Trend detection over the twitter stream. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, pp. 1155–1158. ACM, New York (2010)
15. Pengyi, F., Hui, W., Zhihong, J., Pei, L.: Measurement of Microblogging Network. *Journal of Computer Research and Development* 49(4), 691–699 (2012)

# Social Network Path Analysis Based on HBase

Yan Qiang<sup>1</sup>, Junzuo Lu<sup>1</sup>, Weili Wu<sup>2</sup>, Juanjuan Zhao<sup>1</sup>,  
Xiaolong Zhang<sup>3,1,\*</sup>, Yue Li<sup>1</sup>, and Lidong Wu<sup>2</sup>

<sup>1</sup> College of Computer Science and Technology Taiyuan University of Technology, China  
[{qiangyan,1jz0445,zhaojuanjuan,1y0158}@tyut.edu.cn](mailto:{qiangyan,1jz0445,zhaojuanjuan,1y0158}@tyut.edu.cn)

<sup>2</sup> Department of Computer Science, University of Texas at Dallas, USA  
[{weiliwu,Lidong.wu}@utdallas.edu](mailto:{weiliwu,Lidong.wu}@utdallas.edu)

<sup>3</sup> College of Information Sciences and Technology Pennsylvania State University, USA  
[xiaolong.zhang@gmail.com](mailto:xiaolong.zhang@gmail.com)

**Abstract.** Online social network services have become indispensable in people's daily life. The analysis of data in social network services often involves data mining techniques. However, the quick increase of users in such services posts challenges to develop effective data mining algorithms to deal with large social network data. In this paper, we propose a data-mining algorithm to get the shortest path between nodes in a social network. Based on HBase[1], this algorithm analyzes the social network model, and uses the intermediary degrees and degree central algorithm to optimize the output from cloud platform. With a simulated social network, we validate the efficiency of the algorithm.

**Keywords:** HBase, Parallel BFS, The k-shortest paths, Intermediary degrees.

## 1 Introduction

Social network services (SNS) have influenced our lives by allowing us to interact with each other through virtual connections. However, the analysis of SNS data is not an easy task. In particular, as the number of SNS users increases, the data scale of popular SNS, such as Facebook, posts challenges for effective data analysis. Traditional calculation methods, which are usually designed for small-scale datasets, sometimes cannot handle the computation of large SNS data. For example, in a social network, we often need to compute the shortest path of any two nodes to understand how these two nodes are connected and how close they are. Tradition methods based the relational database model do not work for large social networks, because of complex internal dependency of network nodes. Researchers have proposed some new methods to address this problem; including those using is increasing in data. In order to solve this problem we promote the NOSQL[2] and graph databases. These methods can store the topological structure of a network, which is used to compute the shortest path. Recently, cloud database techniques draw attentions of researchers. Apache

---

\* Corresponding author.

HBase and Google's Bigtable[3] are the representative of cloud databases. HBase is a distributed, open source column-oriented databases and is built on hadoop[4] hdfs distributed storage system. Google's Bigtable is also a structured distributed data storage system, and is based on the Google File System. They both can handle large datasets.

This paper proposes a technique based on HBase. Our algorithm combines data clustering and ranking methods. We first construct a set of paths between any two nodes in a social network, and then rank the paths based on the importance of nodes and user requirements.

## 2 Previous Works

Small world network is also called six degree space theory [5], which argues that although most of nodes in a network are not connected directly, almost all nodes can be reached after a small number of hops. If we see one node in the small world network as a person and the connection represents that people know each other, then the small world network can reflect the small world phenomenon that strangers can be connected by those they mutually know. Small world network was first introduced by Watts and Strogatz[6], who argued small world networks mathematically follow the power law distribution. Barabasi and Albert [7] showed that the power law can be used to simulate small world networks.

Computing the shortest path in social networks often requires effective ways to store network data, such as network-oriented database. Unlike in relational databases in which data is organized as tables, in network-oriented databases data is stored in the network and this storage style allows more agile data query. Neo4j[8] is an example of such databases. With the Neo4j network storage model, algorithms like that by Dijkstra[9] can be used to solve the shortest path problem. However, this kind of approaches strongly depends on hardware, and cannot easily be scaled up for large networks.

NoSQL is also used to deal with social network data. NoSQL is a class of database management system identified by its non-adherence to relational database model. NoSQL database systems have been used by major internet companies, such as Google, Amazon, and Facebook, to deal with huge quantities of data that cannot be handled by traditional relational database systems. To manage large volumes of data, NoSQL do not necessarily follow a fixed schema. Apache HBase and Google Bigtable are examples of NoSQL. While Bigtable is a more proprietary technology, HBase is an open-source, distributed database based on the Hadoop platform. Different from general relational databases, HBase is more suitable to unstructured data.

As the development of cloud computing technology, there appears a variety of corresponding formulations based on cloud platform. Lin et al. [12] have done some research based on the MapReduce framework. They improved the breath-first search (BSF) algorithm in order to let it run in MapReduce distributed formation; it is called the parallel breath-first search. The algorithm can make full use of distributed computing power of the platform for a particular point in the network to get the shortest path

distance to other all points. McCubbin and his colleagues [13] proposed a K-shortest path algorithm to compute the shortest K path by using the sort phase of MapReduce framework.

### 3 Path Algorithm Based on HBase

Our work focuses on transferring traditional shortest path algorithm into a MapReduce format that can be run in the Hadoop cloud platform. Different from the parallel BFS algorithm developed by Lin et al. [12], which cannot make full use of the efficient distributed storage architecture of the Hadoop, our algorithm uses the HBase database and coprocessor in the HBase to improve the operation efficiency of parallel BFS.

#### 3.1 HBase

HBase is a column-oriented database management system that runs on top of Hadoop Distributed File Systems (HDFS). As a non-relational database, HBase does not support a structured query language like SQL.

An HBase system has a set of tables, each of which contains rows and columns. These tables all have a primary key, which is used to access HBase data. In HBase, multiple attributes can be combined together to form column families, and data elements of a column family are then stored together. Thus, when column families are involved in HBase, the table schema and the column families must be predefined and specified.

Social network data can be treated as graph data and stored in HBase. Every edge is associated with a weight. The weight of a path shall be defined to be the sum of the weights of the selected edges. This database stores entries using a 3-tuple of a row, column family, and column qualifier as a key to index values. Graphs are represented as adjacency lists, using the following schema:

```

row := node name
column family := name of the edge type
column qualifier := name of the incident node
value := edge weight (floating point)

```

This graph representation is both straightforward and expressive. It allows the storage of hypergraphs, by simply using different column family identifiers. To indicate which particular graph elements should be considered, we provide our algorithms with a list of edge types as one of their parameter.

To store a shortest path tree for any particular source, we add a column to the HBase table that stores the shortest path tree:

```

row := node name
column family := the string "pointer"
column qualifier := the shortest path source
value := a pointer string

```

Here, the pointer string contains two comma-separated values: a total cost of this shortest path from the node to the source, which we refer to as  $\text{cost}(\text{src}; \text{n})$ , and the next hop in the shortest path to the source.

### 3.2 Parallel BFS to Strike a Single-Source Shortest Path Tree

Based on the HBase method, we implemented our parallel BFS algorithm, and its pseudo-code is as the following:

```

Class Mapper
    Method Map(nid n, node N)
        seen←{}
        For all celli in N do
            Cost = celli.value•cost
            Pointer= celli.value•pointer
            Neighbor= celli.neighbornode
            Seen[pointer]=(neighbor,cost)
            If seen.has(src) and seen.has(dst) then
                totalCost= seen[src].cost+seen[dst].cost
                emit(totalCost, N.key)
        Class Reducer
            visited←{}
            pathList←[]
            Method Reduce(cost,cells)
                For all celli in cells do
                    If emittedPaths<=K then
                        If not visited.has(celli) then
                            Path=ReassemblePath(celli)
                            visited.addAll(path)
                            pathList.append(celli)
                For all pathi in pathList do
                    Emit(pathi)

```

As with Dijkstra's algorithm, we assume a connected, directed graph is stored as adjacency lists. The distance to each node is directly stored alongside the adjacency list of that node, and initialized to 1 for all nodes except for the source node. In the pseudo-code, we use  $n$  to represent the node id (an integer) and  $N$  to denote the node's corresponding data structure (adjacency list and current distance). The algorithm works by mapping all nodes and emitting the key-value pair for each neighbor on the node's adjacency list. The key contains the node id of the neighbor, and the value saves the current distance to the node plus one. If we can reach node  $n$  with distance  $d$ , then we must be able to reach all the nodes that are connected to  $n$  with distance  $d + 1$ . After shuffling and sorting, the reducer stage will receive keys corresponding to the destination node ids and distances corresponding to all paths leading to that node. The reducer will select the shortest value of these distances and then update the distance in the node data structure.

It is apparent that parallel BFS is an iterative algorithm, where each iteration corresponds to a MapReduce job. At the first time to run the algorithm, those nodes that are connected to the source can be identified. In the second iteration, nodes that are connected to those nodes identified in the first run can be discovered. As iteration keeps going on, the algorithm expands the search frontier, and eventually all nodes can be discovered with their shortest distances.

The BFS method requires both map and reduce processes, as well as the ampreduce process to pass graph structure. These processes becomes a burden to computational efficiency, so we offer improvements to BFS by leveraging HBase coprocessors.

Coprocessors can be loaded globally on all tables and regions hosted by the region server, and these are known as system coprocessors. The administrator can also specify which coprocessors should be loaded on all regions for a table on a per-table basis, and these are known as table coprocessors.

We use the coprocessor to improve the BFS method. The map phase is responsible for the value of the transmission path and the network structure in the traditional BFS method, but Coprocessor component architecture allows us to complete the update of the value of the node shortest path  $d$  in the map phase. This method has two advantages. First, it only needs the map phase, so a lot of the time consumed in the process of sort-shuffle can be saved. Second, the data storage mode of HBase can help to reduce the consumption of the data transfer process. By doing so, updates to the pointer tree are made to some nodes before the mappers read their data. We refer to this property as pointer cascading. This leads to one iteration of the algorithm that can extend the effective frontier of the shortest paths by more than one hop. We can force HBase to updates the data after map to reduce the number of iterations.

### 3.3 K-Shortest Paths

For our application, we hope to find paths that are cycle free and also meet the additional requirement: each path in the set of k-shortest paths contains a node not presented on any other path in the set. In this way, we can find new nodes. This property is called node uniqueness. After the single-source shortest path tree that is stored in the HBase database got from our algorithm in Section 3.2, we can determine the k-shortest paths in the graph with one additional pass. In our Map function, we check each node to see if it has a shortest path pointer to both the source and destination. If the answer is yes, the path from the source through the node to the destination formed by pasting the two shortest paths together is a candidate for a k-shortest path from the source to destination. To inform the reducer of this fact, we emit a key/value pair in which the key is the sum of both pointers' costs and the value is the node's identifier. We can reverse one of the two shortest paths, because we assume that the graph is undirected. Courtesy of the sort-shuffle phase, our reducer receives the candidate shortest paths in cost order. Ties are broken by the node numbering; the lowest numbered paths are discovered first. The reducer must, however, do more than simply choosing the first resultant  $k$  candidates. We are interested in unique shortest paths - paths that have at least one "interesting node". Therefore our reducer keeps a set of nodes that have seen on previous paths it accepts, and rejects the shortest path

suggestions that are just composed of the nodes it has seen before. The reducer must also guard against cycles that may be produced by the shortest path pointer algorithm. In the omitted function ReassemblePath, we reconstruct each proposed path from HBase and check it for cycles before accepting it.

The pseudo-code of this process is as the following:

```

Class Mapper
    Method Map(nid n, node N)
        seen←{}
        For all celli in N do
            Cost = celli•value•cost
            Pointer= celli•value•pointer
            Neighbor= celli•neighbornode
            Seen[pointer]=(neighbor,cost)
            If seen.has(src) and seen.has(dst) then
                totalCost=seen[src].cost+seen[dst].cost
                emit(totalCost, N.key)
    Class Reducer
        visited←{}
        pathList←[]
        Method Reduce(cost,cells)
            For all celli in cells do
                If emittedPaths<=K then
                    If not visited.has(celli) then
                        Path=ReassemblePath(celli)
                        visited.addAll(path)
                        pathList.append(celli)
            For all pathi in pathList do
                Emit(pathi)

```

### 3.4 Optimization Sorting

We get the K source node to the destination node of the shortest path set in the previous step from the cloud platform. A user can get paths through the cloud platform to look at how she is connected with others. However, can the user easily read path centralized information?

The answer is no, because the path ordering set is just according to the path value size, and no other information is available. When calculating the path, if the value of K is too small, many important paths may be missed; if K is too big, there might be too many paths. Thus, it is important to choose an optimal value for K so that an appropriate number of paths can be identified.

There are many methods to optimize network diagram path. According to the needs of users, we can use different combination algorithm. Here we choose intermediary degree and degree of central algorithm to optimize cloud platform output results.

In the real case, the user not only expects to know who is in the path, but also needs to know if someone in the relationship chain plays a key role, because the key role of people may be the one who is known by both parties. So we choose intermediary degrees (betweenness) to optimize the path set.

Betweenness Centrality, betweenness for short [14], is an important concept in social network analysis. The betweenness of one node represents the number of the shortest path of all the nodes. Betweenness is a good description of the contribution of those nodes in every path to the connection between two nodes. The larger betweenness of a node is, the more paths go through the node and the more probable the node connects other nodes.

Consider a weighted directed (multi)-graph  $G = (V, E)$  with  $n = |V|$ ,  $m = |E|$ . Let  $SP_{st}$  denote the set of shortest paths between source  $s$  and target  $t$  and  $SP_{st}(v)$  the subset of  $SP_{st}$  consisting of paths that have  $v$  in their interior. Then, the betweenness centrality for node  $v$  is

$$C_B(v) = \sum_{s \neq v \neq t \in V} \sigma_{st}(v) / \sigma_{st} \quad (1)$$

Where  $\sigma_{st} := |SP_{st}|$  and  $\sigma_{st}(v) := |SP_{st}(v)|$ .

In addition we choose degree centricity (degree centrality) as another optimization index. The degree centrality of a node is defined as the number of edges that connect to this node [15]. Usually, the higher the centrality of a node is, the more popular or well-connected the node is. In our algorithm, the degree centrality for node  $v$  is written as:

$$C_D(v) = \deg(v) / (n - 1) \quad (2)$$

Where  $\deg(v)$  is the degree of  $v$  and  $n$  is the number of  $v$ . The algorithm has low computing complexity and computing resources consumption.

## 4 The Analysis of the Experiment

We conducted an experiment to evaluate the performance of our algorithm.

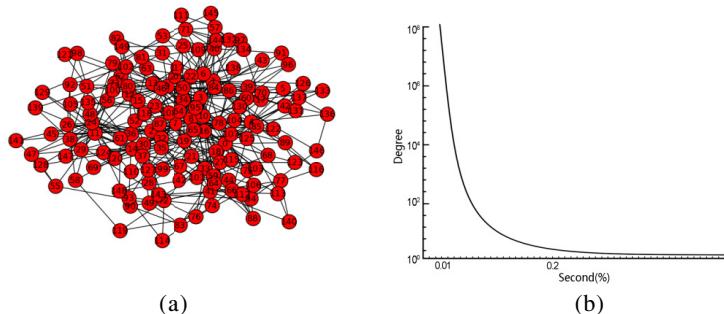
### 4.1 Hardware Configuration

In order to verify the correctness of the algorithm, we used Hadoop for cloud platform configuration. There are 10 nodes in this cloud cluster for testing, and each node was configured to use a 2.6-GHz CPU, 8GB of RAM, and 1TB of hard disk. We have 128 mapper and 64 reduce in default. After large number of tests, we set the rate between the quantity of mapper, reduce and the running speed of image algorithm to 2:1.

### 4.2 Network Model

It is hard to get the real social network data to test. Thus, we used mathematic modeling to simulate a social network based on the model proposed by Holme[15].

The network follows a small-world network model and obeys the power-law distribution. We used Networkx, a software package written with Python, to generate the network data. Figure 1 (a) shows the network pattern of the simulated network, and Figure 1 (b) illustrates the distribution of node degree of the simulated network.



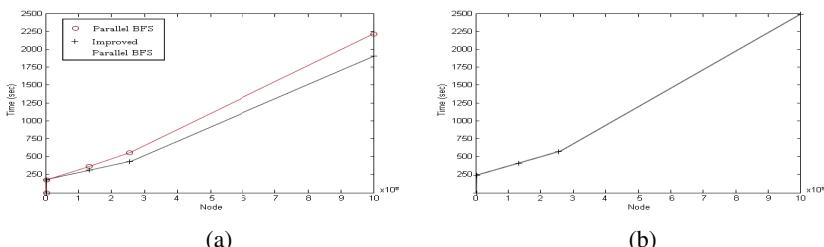
**Fig. 1.** (a) one example “small world” network. (b) probability distribution of simulated network node

## 5 Results and Discussion

### 5.1 Time Complexity

In theory, for each network diagram which complexity of node degree is  $O(1)$ , its time complexity is  $O(|\log |E||)$ . This is because that each line will receive a new message in the process of iterative formulation, and the amount of all the message is  $O(E)$ . Then, the complexity of storing data to HBase table structure is  $O(\log |E|)$ .

Figure 2 (a) compares the efficiency of our algorithm with the parallel BFS. As shown, our algorithm outperforms the traditional parallel BFS.



**Fig. 2.** (a) Efficiency Comparison of two algorithms. (b) Running time of k-shortest paths algorithm of different nodes

The time complexity of K-shortest path algorithm and BFS algorithm are the same. It is  $O(V \log |V|)$ , and the time complexity in shuffle-sort phase of mapreduce is  $O(E \log |E|)$ . Figure 2 (b) shows the running time of k-shortest paths as the function of node number.

## 5.2 Discussion

Figure 5 compares the number of BFS iteration with different numbers of nodes. As shown, the impact of cascade phenomenon on the number of iteration is significant, in particular when data is forced to be written into HBase immediately after map.

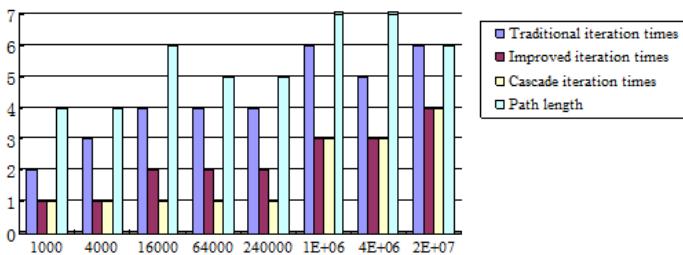


Fig. 3. iterations for breadth first search

Finally we tested our program on a huge network, which contains one hundred million nodes and six hundred million edges. The running time to get the shortest paths with our method is 6 hours and 30 minutes, which is acceptable. Half of the total time was used for parallel BFS algorithm, and the other half was for running K-shortest path algorithm. We set K to be 20, but the running time of the optimization algorithm was neglected, compared with parallel BFS and K-shortest algorithms.

## 6 Conclusion

In this paper, we proposed a parallel BFS algorithm to compute the shortest path of a social network based on the HBase. This algorithm analyzes the social network model, and uses the intermediary degrees and degree central algorithm to optimize the output from cloud platform. With a simulated social network, we validate the efficiency of the algorithm. The experiment results indicate that our algorithm can improve the efficiency of parallel BSF.

**Acknowledgements.** This study was supported by the National Natural Science Foundation of China (Grant No. 61202163, 61240035); Natural Science Foundation of Shanxi Province (Grant No. 2012011015-1) and Programs for Science and Technology Development of Shanxi Province (Grant No. 20120313032-3). This work was also supported in part by the US National Science Foundation (NSF) under Grant no. CNS-1016320 and CCF-0829993.

## References

1. HBase Homepage, <http://hbase.apache.org>
2. Lakshman, A., Malik, P.: Cassandra: A Decentralized Structured Storage System. SIGOPS (2010)
3. Chang, F., et al.: Bigtable: A Distributed Storage System for Structured Data. In: OSDI (2006)
4. Hadoop, <http://hadoop.apache.org>
5. Missem, M.M.S.: The small world of web network graphs. In: International Multi Topic Conference on Wireless Networks, Information Processing and Systems, IMTIC (2008)
6. Watts, D.J., Strogatz, S.H.: Collective dynamics of ‘small-world’ networks. *Nature* 393, 440 (1998)
7. Barabási, A.-L., Albert, R.: Emergence of scaling in random networks. *Science* 286, 509 (1999); Barabási, A.-L., Albert, R., Jeong, H.: Mean-field theory for scale-free random networks. *Physica A* 272, 173 (1999)
8. Hoque, I., Gupta, I.: Disk Layout Techniques for Online Social Network Data. In: IEEE Internet Computing (2012)
9. Dijkstra, E.W.: A note on two problems in connexion with graphs. *Numerische Mathematik*, 269–271 (1959)
10. The Apache Software Foundation. The Hadoop Distributed File System: Architecture and Design, <http://hadoop.apache.org/>
11. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. *Commun. ACM* 51(1), 107–113 (2008)
12. Lin, J., Dyer, C.: Data-Intensive Text Processing with MapReduce. *Synthesis Lectures on Human Language Technologies*. Morgan & Claypool Publishers (2010)
13. McCubbin, C., Perozzi, B.: Finding the ‘Needle’: Locating Interesting Nodes Using the K-Shortest Paths Algorithm in MapReduce. In: 2011 11th IEEE International Conference on Data Mining Workshops (2011)
14. Brandes, U.: A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology* 25(2), 163–177 (2001)
15. Qin, L., Li, H.: Centrality analysis of BBS reply networks. In: 2011 International Conference on Information Technology, Computer Engineering and Management Sciences, ICM 2011, September 24-25 (2011)
16. Holme, P., Kim, B.J.: Growing scale free networks with tunable clustering. *Physical Review E* 65 (2002)

# Community Expansion Model Based on Charged System Theory

Yuanjun Bi<sup>1</sup>, Weili Wu<sup>1,2</sup>, Ailian Wang<sup>2</sup>, and Lidan Fan<sup>1</sup>

<sup>1</sup> Department of Computer Science, University of Texas at Dallas  
Richardson, TX 75080, USA

<sup>2</sup> College of Computer Science and Technology, TaiYuan University of Technology  
Taiyuan, Shanxi 030024, China  
{yuanjun.bi,weiliwu,lidan.fan}@utdallas.edu, ym4008cn@yahoo.com.cn

**Abstract.** Recently, the phenomenon of influence propagation becomes a hot topic in social networks. However, few existing influence models study the influence from communities, which has a large range of applications. In this paper, we use the charged system model to represent the social influence. This model provides a natural description about the behaviors of influence and explains why the influence makes communities expand. Based on this physical model, we propose two objective functions for choosing proper candidates to enlarge a community, considering of the cost and benefit issue. Then a linear programming approach is given to maximize those two objective functions. We validate our ideas and algorithm using two real-world networks. The results demonstrate that our model can choose excellent propagation candidates for a specific community, comparing to other two algorithms.

**Keywords:** community expansion, physical model, social influence, linear programming.

## 1 Introduction

Social influence has many sources such as an individual, organizations and society in general [1]. They changed the behaviors of people around them. For example, they influence the customers' attitude towards products or electors' decision towards political candidates. In macroscopic view, the results of influence lead to the community evolution including expansion, contraction and no change [2].

Due to the popular using of online social media such as Facebook and Twitter, more and more researchers are interested in the influence through "word of mouth". They build several models to represent the progress of influence spreading and aim to find algorithms to maximize or minimize the influence [3,4,5,6]. Some researchers study the laws of community diffusion and try to find approaches to fit changed community model over time [2,7]. However, few work study on the influence coming from a community which can provide a proper

---

\* This work was supported in part by the US National Science Foundation (NSF) under Grant no. CNS-1016320 and CCF-0829993.

strategy to maximize the community size. In fact, the problem of community expansion has a large range of applications: Every company is looking for more customers to increase new sales. How to find potential customers is important especially when the marketing budget is limited. For political candidates, enlarging alliance is a good approach to spread their influence towards decision making process. Modeling such influence based on community structure offers valuable insight in choices during campaign activities. Trade shows and exhibitions organizers also want to choose excellent participators, such that the fame of exhibition can be improved. All these real world issues need to model the influence from a group and provide a strategy to enlarge the group.

In this paper, we build a model based on charged system, a physical model to represent the phenomena that how the influence power makes communities evolve. In the marketing scenario, each company holds its clients because of the influence from that company. However, some clients will change to accept another company's service if they received considerable promotion. In another word, an individual will join another community if the influence comes from that community is big enough. While in charged system, an electric charge will move if the electric force from a specific electric field breaks the stable status. Since there exists similarities between social phenomena and physical phenomena, we want to utilize the charged system to describe the progress that a community provides promotion to attract new members.

The challenged parts are how to describe the features in social network using characters in charged system. And which electric charge should be chosen so that there are more electric charges will be attracted but with less additional force. What we should notice here is, once a single electric object is attracted to move, the global stable status will be broken and other electric charges will move as well. That is similar as the influence result of "word of mouth". People change their behavior because of their friends. In this paper we employ Coulomb law to simulate the attraction force and propose objective functions which try to minimize expansion cost and maximize the size of new community members.

The main contributions of this paper include: i)Employ charged system to build model for describing features in social network and the progress of community expansion. ii)Use Coulomb law to build objective functions for expanding the community. iii)Use linear programming to maximize two objective functions and do the experiments to validate the idea base on two real world datasets.

The rest of this paper is organized as follows. In the next section, a brief overview of related work is introduced. Section 3 model the community expansion progress using charged system theory. A linear programming approach based on the models is proposed in section 4. The simulation results and conclusions are showed in section 5 and 6 respectively.

## 2 Related Work

There is much research on community evolution [2,8,9]. Aggarwal and Yu [2] named three transitions of a community: expansion, contraction and no change.

They put a weighting scheme on edges according to interaction changes for detecting the community evolution. Nguyen *et al.* [8] analysis four basic events occur in dynamic network: adding or removing nodes and edges. They propose adaptive algorithms separately to update the network community structure. Chakrabarti *et al.* build objective function to monitor clustering over time. Their optimization methods based on the idea that the cost of capturing snapshot and temporal clustering stream should be minimized.

An abundance of work focuses on influence maximization problem [3,4,5,10]. Kempe [3] gave the definition of that problem which is to find a set of initial set of users in a social network such that from this set the spread of influence in the network can be maximized. Linear Threshold (*LT*) Model and Independent Cascade(*IC*) Model are two main approaches to formalize the influence maximization problem. Chen, *et al.* [4] propose a *MIA* model and its heuristic algorithm to address the scalability and efficiency issue in large scale networks. Direct marketing is an important application of social influence [11,12,13,14]. The influence from friends usually changes people's behavior. Domingos *et al.* [11] compute the customer's probability of buying based on Markov random fields. They value the customer's benefit according to their trade history combined with the discount cost of offering to them. Hartline *et al.* [15] propose the influence-and-exploit strategy to trying to find optimal marketing strategy. In influence step the sellers offer free products while in exploit step the sellers fix the price at a optimal price. Tao *et al.* [6] proposed TABI, a heuristic algorithm to solve the participation maximization problem. This algorithm calculates participants' influence and allocate thread according to influence ranking. Their method accounts in weights between the individual and its neighbors. However, TABI only considers people who have participated in the forum, which means the algorithm only chooses candidates from people who have connections with the community.

### 3 Problem Formulation

In this section, we will introduce the physical charged system firstly. Then we will transfrom the features in social network into the characters in charged system. Coulomb law is used for formulating the progress of attracting new members. Finally objective functions will be proposed to give an optimal strategy for community expansion.

#### 3.1 Electric Field and Charged System

In physics, a single charge point can create an electric field which exerts force on other charged objects around without touching. This electric force makes a charge particle move closer or further to another. Coulomb gave experiments which showed that the electric force between two charged points  $i, j$  is proportional to the product of quantity on the two electric particles  $q_i, q_j$  and inversely

proportional to the square of separation distance between them[16]. Function 1 shows the computing of electric force between two charges.

$$\mathbf{F}_{ij} = h \frac{q_i q_j}{r_{ij}^2} \quad (1)$$

where  $h$  is the Coulomb constant and  $r_{ij}$  is the distance between charges  $i$  and  $j$ .  $\mathbf{F}_{ij}$  is a vector quantity. It appears attractive if the charges are of opposite sign and repulsive otherwise.

### 3.2 Presentation of Social Influence

In this part, we want to transform features in social influence into variables in electric field theory. Electric field theory is chosen because electric force among charges causes the moving of these particles, which is similar as the phenomena in social network that social influence from different communities change people's behaviors.

**Electric Charges.** We see an individual in social network as a charged particle. The individual may do not belong to any community, or belong to several communities in social network, which means communities are overlapping. The quantity of a particle  $q$  refers to the attractiveness to a community  $C$  when  $C$  wants to choose "seeds" to expand itself. Since in real world, people known by more people are more likely to be celebrities who have much more influence towards others, we define the degree of the node as its charged quantity property in Function 2.

$$q_i = n_i \quad (2)$$

where  $n_i$  is the degree of the node  $i$ .

For a specific community which we want to expand its size, we call it *Target Community*(TC). TC can be seen as a large electric particle which contains all the nodes that in TC. To compute the electrical quantity of TC, we should not only consider the number of nodes in TC but also the connection density inside the TC. A community which has more members and tighter density connection seems more powerful to nodes outside TC.

$$Q_{TC} = N_{TC} * Den_{TC} \quad (3)$$

Function 3 gives the formula about TC's electrical quantity, where  $N_{TC}$  is the number of the nodes inside TC.  $Den_{TC}$  is the connection density inside TC. We set the electrical polarity of an TC as positive. For the particle  $i$  outside TC, we set its electrical polarity as negative. As a result, TC has attractive force towards particles outside TC.

**Distance.** An important variable in Function 1 is the distance  $r$ . The distance  $r_{ij}$  between two particles  $i$  and  $j$  refers to the shortest path length between them. However, the distance between a particle  $i$  and the community  $C$ ,  $r_{iC}$ , should

consider other features in addition to the shortest path length between  $i$  and any vertex in  $C$ . Function 4 gives the definition.

$$r_{iC} = \frac{dis_i}{\log(m_i + 1) + 1} \quad (4)$$

where  $dis$  means the shortest path length from  $i$  to all vertexes in  $C$ ,  $m_i$  is the number of  $i$ 's neighbors who are already in the specified community  $C$ . In the study of [17], the experiments show that the main feature to affect an individual to join a community is the the number of friends in a community. However, the relation between the probability of joining a community and the number of friends in that community is under the “law of diminishing returns”. We use  $\log(m + 1)$  to indicate that individual who is more likely to join in  $C$  appears more closer to  $C$ .  $m + 1$  makes the function keep physical meaning when  $m = 0$ . Another 1 guarantees the distance is bigger than 0.

**Attractive Force.** Recall that particles which do not belong to  $TC$  are seen as negative. From Function 1, we can compute the attractive force from  $TC$  to a certain particle  $i$  by Function 5,

$$F_{TC \rightarrow i} = h * \frac{Q_{TC} * n_i}{r_{iTC}^2} \quad (5)$$

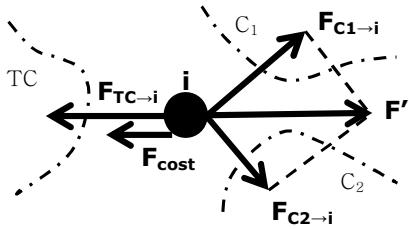
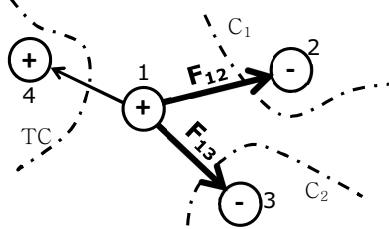
where  $h$  is the Coulomb constant,  $Q_{TC}$  is the electrical quantity of  $TC$ ,  $n_i$  is the  $i$ 's electrical quantity,  $r_{iTC}$  is the distance between  $i$  and  $TC$ .

### 3.3 Expansion Progress and Objective Function

In this section we use electric field theory to explain the progress of community expansion. Objective functions will be built based on the goals of maximizing the benefit while minimizing the cost. We will illustrate the community expansion step by step.

**Initial State.** We can see the snapshot of social network at a given time point as the initial state. For a vertex  $i$ , as shown in Figure 1, it receives the electric force exerted by  $TC$ ,  $F_{TC \rightarrow i}$ , and the resultant force exerted by other communities except  $TC$ ,  $F'$ . From the view of  $TC$ ,  $F'$  has the opposite direction with  $F_{TC \rightarrow i}$ .  $i$  will not belong to  $TC$  if  $F_{TC \rightarrow i}$  is smaller than a threshold. Recall that communities are allowed to be overlapped, so that  $i$  can belong to different communities.

**Promotion State.** In order to attract new members to join in  $TC$ ,  $TC$  should add additional force towards vertexes outside  $TC$  to make them move. This additional force can be seen as promotion discount to new members. As illustrated in Figure 1,  $F_{cost}$  should be minimized when the promotion budget is limited. Suppose the discount cost on every vertex is the same. Then we should choose vertexes to which  $TC$  appears more attractive such that the less  $F_{cost}$  will make those vertexes move. Suppose the promotion budget allows  $TC$  to select  $K$

**Fig. 1.** Force analysis**Fig. 2.** Influence to neighbors

vertexes to obtain discount, the vertexes set of the whole network is  $V$ , then we should select proper vertex set  $S$  in  $\zeta$  to make the value of Function 6 maximum.

$$f_1(S) = \sum_{i \in S} \mathbf{F}_{TC \rightarrow i} = \sum_{i \in S} h \frac{q_i * Q_{TC}}{r_{iTC}^2} \quad (6)$$

where  $S \in \zeta$ .  $\zeta$  contains all possible combination sets which have  $K$  vertexes outside  $TC$ . That is

$$\zeta = \{S \mid S \subset V \setminus V_{TC}, |S| = K\}$$

$q_i$  can be obtained by Function 2 and  $r_{iTC}$  can be computed by Function 4. The electrical quantity  $Q_{TC}$  keeps the same during the promotion state. Function 7 gives the first objective function for choosing vertexes which can reduce the cost.

$$\max f_1(S) = \max_{S \in \zeta} \sum_{i \in S} \frac{q_i}{r_{iTC}^2} \quad (7)$$

**Expansion State.** After promotion activity, some vertexes will move to  $TC$  and their behaviors will affect their neighbors. To maximize the benefit, the vertexes which have more influence to neighbors are better choice. Suppose once a vertex outside  $TC$  decides to join in  $TC$ , its polarity becomes positive. Then it will exert attractive force to its neighbors who are not in  $TC$ . Figure 2 shows that vertex 1 has attractive force to vertex 2 and 3 when it decided to join in  $TC$ . These attractive force will increase the probabilities of 2 and 3 moving to  $TC$ . We hope the coming of vertex 1 can make its neighbors obtain more attractive force from  $TC$ . The benefit function is given as Function 8,

$$f_2(S) = \sum_{i \in S} \sum_{j \in W_i} F_{TC \rightarrow j}^* - F_{TC \rightarrow j} \quad (8)$$

where  $S \in \zeta$ .  $W_i$  contain  $i$ 's neighbors who are not in  $TC$ .  $F_{TC \rightarrow j}^*$  is the attractive force from  $TC$  to  $j$  after  $j$ 's neighbor  $i$  came to  $TC$ . From Function 3 we can compute  $F_{TC \rightarrow j}^*$ . Since  $q_j$  keeps the same value. Function 8 equals to Function 9, which aims to choose individuals who have bigger influence to others.

$$\max f_2(S) = \max_{S \in \zeta} \sum_{i \in S} \sum_{j \in W_i} \left( \frac{Q_{TC}^*}{(r_{jTC}^*)^2} - \frac{Q_{TC}}{r_{jTC}^2} \right) \quad (9)$$

**Algorithm 1.** TCE Algorithm

- 
- Input:**  $G = (V, E), TC, K, \alpha$ ;  
**Output:** promotion seed set  $S$ ;
- 1  $S = \emptyset$ ;
  - 2 Compute  $Q_{TC} = N_{TC} * Den_{TC}$ ;
  - 3 For each  $x_i \in V \setminus V_{TC}$ , compute  $F_{TC \rightarrow i} = h \frac{q_i * Q_{TC}}{r_{TC}^2} = h \frac{n_i * Q_{TC} * (\log(m_i+1)+1)^2}{dis_i^2}$ ;
  - 4 Compute the sum force of  $x_i$ 's neighbors,  $\sum_{j \in W_i} F_{TC \rightarrow j}$ ;
  - 5 Assume  $x_i$  was in  $TC$ , recompute the electrical quantity of  $TC$   $Q_{TC}^*$  and the sum force of  $x_i$ 's neighbors,  $\sum_{j \in W_i} F_{TC \rightarrow j}^*$ ;
  - 6 Compute  $c_i = \alpha F_{TC \rightarrow i} + (1 - \alpha)(\sum_{j \in W_i} F_{TC \rightarrow j}^* - \sum_{j \in W_i} F_{TC \rightarrow j})$  for each  $x_i$ ;
  - 7 Call linear programming using Function 11 to obtain  $S$ ;
- 

$Q_{TC}^*$  and  $r_{jTC}^*$  is the updated value when  $i$  joined into  $TC$ . Our goal is to choose proper  $S$  such that Functions 7 and 9 can be satisfied at the same time.

## 4 A Linear Programming Approach

Two objective functions, Function 7 and Function 9 reflect two aspects of strategy for expanding community  $TC$ : cost and benefit. To find proper strategy under different requirements, a parameter  $\alpha$  are used to adjust the weight of each objective functions.

$$f(S) = \alpha f_1(S) + (1 - \alpha) f_2(S) \quad (10)$$

The parameter  $\alpha$  expresses the importance of cost requirement in the expanding strategy. It ranges between zero and one, hence Function 10 can be tuned to cost model( $\alpha$  is close to one) or benefit model( $\alpha$  is close to zero).

Let  $G = (V, E)$  be an undirected graph and  $G$  is departed to several overlapped communities. One of these communities is  $TC$ . We define binary variable  $x_i = 1$  if vertex  $x_i$  is selected in optimal solution for expanding  $TC$ , otherwise  $x_i = 0$ . Note that the selected  $x_i$  must be nodes outside  $TC$ . Let  $c_i$  be the value of Function 10 when  $S = \{i\}$ . Then the Target Community Expansion problem(TCE) can be formulated as:

$$\text{Maximize} \sum_{i \in V \setminus V_{TC}} c_i x_i \quad (11)$$

$$\text{subject to} \sum_{i \in V \setminus V_{TC}} x_i = K \quad (12)$$

$$x_i \in \{0, 1\}, \forall x_i \in V \setminus V_{TC} \quad (13)$$

The objective Function 11 combines Function 7 and Function 9. Constraints 12 enforces that the promotion cost is under the budget limitation, which means only  $K$  nodes are chosen to accept promotion.  $K$  is an input parameter. Constraints 13 enforces binary restrictions on the x-variables. The TCE algorithm represents the whole progress of  $TC$  expanding progress.

## 5 Experiment

In this section, we conduct experiments on TCE algorithm as well as other two algorithms on two real-world networks. There are two aspects to measure our experiments' performance: (1) Its capacity of attracting new members; (2) The effect to TC after these selected nodes joining TC.

### 5.1 Experiment Setup

**Datasets.** We use two realistic data sets: American College Football and Arenas Email. *American College Football(ACF)*, which has 115 nodes and 1226 edges, is a representation of the schedule of Division I games for the 2000 season, in which vertices represent teams(identified by their college names) and edges are regular-season games between the two teams they connect. *Arenas/email* comes from email interchange network, Univ. of Rovira i Virgili, Tarragona. The nodes are the members in this university and the edges represent email interchanges between members. It has 1133 nodes and 10903 edges.

**Algorithms.** To find  $TC$  and other communities, we first partition the social network graph into several communities using the methods in [18]. The community with the maximum size is chosen as the Target Community  $TC$ . Our TCE algorithm uses Matlab linear optimization function to obtain those K nodes. We compare TCE with other two algorithms. (1)Random: As a baseline comparison, simply select K nodes from communities other than  $TC$  randomly. (2)MaxDegree: A greedy algorithm which select K nodes outside  $TC$  which have the maximum node degree. We run the simulation 1000 times and take the average of results.

### 5.2 Experimental Results

**Capacity of Attracting New Customers.** We measure the capacity of attracting new customers by counting the number of newcomers(NC) under a certain threshold. For the selected K nodes from above three algorithms, we compute the attractive force from  $TC$  to that node using Function 5. If the attractive force is greater than the threshold we consider that node will join in  $TC$ . NC is the number of nodes that accepted the promotion from  $TC$ ( $NC \leq K$ ). We do the experiments under different K value and  $\alpha$  value. In Function 10,  $\alpha(0 \leq \alpha \leq 1)$  adjusts the weight between objective Function 7 and objective Function 9.

Figure 3 and Figure 4 show the number of newcomers on ACF/Team dataset under different K value and  $\alpha$  value. First we tune the K value. TCE always works better than other two algorithm. For a specific  $\alpha$  value 0.5 , TCE is 40%, 50% better than RANDOM, MaxDegree respectively when  $K = 25$ . While for tuning  $\alpha$  value, TCE is 80%, 60% better than RANDOM, MaxDegree respectively when  $\alpha = 0.7$ . For the moderate sized graph Arenas Email, as showed in Figure 6 and Figure 7, our TCE performs best on different K and  $\alpha$  value. When

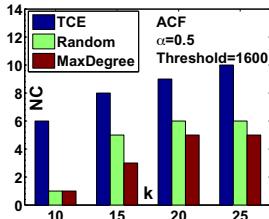


Fig. 3. NC/ACF/K

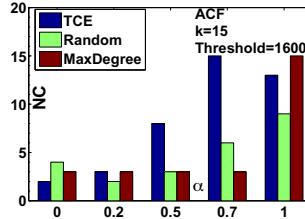


Fig. 4. NC/ACF/alpha

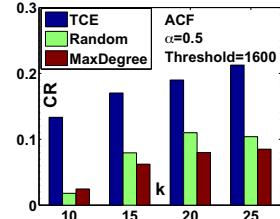


Fig. 5. CR/ACF/K

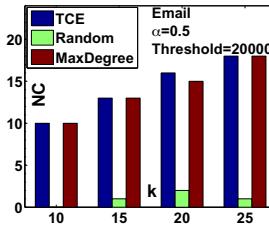


Fig. 6. NC/Email/K

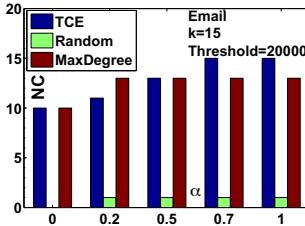


Fig. 7. NC/Email/alpha

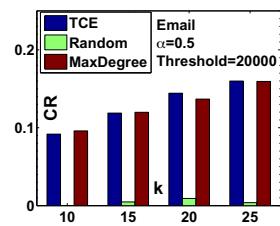


Fig. 8. CR/Email/K

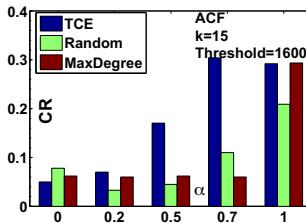


Fig. 9. CR/ACF/alpha

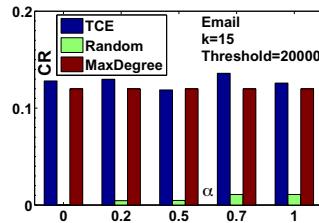


Fig. 10. CR/Email/alpha

$K = 20$ , for a specific  $\alpha = 0.5$ , TCE is 87.5%, 6.25% better, while for a specific  $K = 15$ , TCE is 93.3%, 13.3% better when  $\alpha = 0.7$ , comparing to RANDOM and MaxDegree respectively. Generally, TCE has better capacity of attracting newcomers no matter how  $K$  or  $\alpha$  changes. Specially, TCE's performance becomes better when the  $\alpha$  value arises, which means the first objective function can attract more newcomers. RANDOM works worse in Arenas Email than in ACF, because Arenas Email has much more nodes, which makes RANDOM has higher probability to choose bad nodes.

**Effect to TC Structure.** The effect of newcomers to  $TC$  is another important evaluation criterion, especially when the community considers doing expansion activities next time. We hope those newcomers can make  $TC$  has greater ability to attract new members. Here we define *Cut Ratio* as the measurement of attracting newcomers ability.  $CR = Cut/P_{out}$ , where  $Cut$  is the number of edges which has one node in  $TC$  and another node outside  $TC$ .  $P_{out}$  is the number of

edges that totally outside  $TC$ .  $CR$  represents the ratio between “open links” of  $TC$  and links that are not in  $TC$ . If the coming of selected nodes can increase  $CR$  value, we say that they have good effect to  $TC$  structure. In Figure 5 and Figure 8-10, we record the difference value of  $CR$  after the selected nodes from three algorithms joining  $TC$ . we can see that our TCE algorithm has higher  $CR$  difference value than RANDOM and MaxDegree under different  $K$  and  $\alpha$  on both two datasets. That means the nodes chosen from our algorithm have higher ability to help  $TC$  attract new members in the future.

## 6 Conclusions

In this paper, we used physical charged system model to represent the influence from communities in social network. Two objective functions were proposed considering of cost and benefit issue in the community expanding progress. Then a linear programming was given to choose proper nodes for a Target Community. Experiment results based on two real world datasets show that our algorithms can attract more high value newcomers for a specific community.

## References

1. Aggarwal, C.C.: Social Network Data Analytics. Springer (2011)
2. Aggarwal, C.C., Yu, P.S.: Online analysis of community evolution in data streams. In: SDM (2005)
3. Kempe, D., Kleinberg, J., Tardos, E.: Maximizing the spread of influence through a social network. In: KDD (2003)
4. Wang, C., Chen, W., Wang, Y.: Scalable influence maximization for independent cascade model in large-scale social networks. In: Data Mining and Knowledge Discovery (2012)
5. Saito, K., Nakano, R., Kimura, M.: Prediction of information diffusion probabilities for independent cascade model. In: Lovrek, I., Howlett, R.J., Jain, L.C. (eds.) KES 2008, Part III. LNCS (LNAI), vol. 5179, pp. 67–75. Springer, Heidelberg (2008)
6. Sun, T., Chen, W., Liu, Z., Wang, Y., Sun, X., Zhang, M., Lin, C.: Participation maximization based on social influence in online discussion forums. In: ICWSM (2011)
7. Leskovec, J., Backstrom, L., Kumar, R., Tomkins, A.: Microscopic evolution of social networks. In: KDD (2008)
8. Nguyen, N.P., Dinh, T.N., Xuan, Y., Thai, M.T.: Adaptive algorithms for detecting community structure in dynamic social networks. In: INFOCOM (2011)
9. Chakrabarti, D., Kumar, R., Tomkins, A.: Evolutionary clustering. In: KDD (2006)
10. Goyal, A., Bonchi, F., Lakshmanan, L.V.S.: Learning influence probabilities in social networks. In: WSDM (2010)
11. Domingos, P., Richardson, M.: Mining the network value of customers. In: KDD (2001)
12. Leskovec, J., Adamic, L., Huberman, B.: The dynamics of viral marketing. ACM Transactions on the Web (2007)
13. Richardson, M., Domingos, P.: Mining knowledge-sharing sites for viral marketing. In: KDD (2002)

14. Shimp, T.A.: Advertising promotion: Supplemental aspects of integrated marketing communications. South-Western College Pub. (2002)
15. Hartline, J., Mirrokni, V., Sundararajan, M.: Optimal marketing strategies over social networks. In: WWW (2008)
16. Halliday, D., Resnick, R., Walker, J.: Fundamentals of Physics, 8th edn. Wiley (2007)
17. Backstrom, L., Huttenlocher, D., Kleinberg, J., Lan, X.: Group formation in large social networks: membership, growth, and evolution. In: KDD (2006)
18. Hu, Y., Chen, H., Zhang, P., Di, Z., Li, M., Fan, Y.: Comparative definition of community and corresponding identifying algorithm. Phys. Rev. (2008)

# Centrality and Spectral Radius in Dynamic Communication Networks

Danica Vukadinović Greetham, Zhivko Stoyanov, and Peter Grindrod

Centre for the Mathematics of Human Behaviour

Department of Mathematics and Statistics

University of Reading, UK

{d.v.greetham,z.v.stoyanov,p.grindrod}@reading.ac.uk

**Abstract.** We explore the influence of the choice of attenuation factor on Katz centrality indices for evolving communication networks. For given snapshots of a network observed over a period of time, recently developed communicability indices aim to identify best broadcasters and listeners in the network. In this article, we looked into the sensitivity of communicability indices on the attenuation factor constraint, in relation to spectral radius (the largest eigenvalue) of the network at any point in time and its computation in the case of large networks. We proposed relaxed communicability measures where the spectral radius bound on attenuation factor is relaxed and the adjacency matrix is normalised in order to maintain the convergence of the measure. Using a vitality based measure of both standard and relaxed communicability indices we looked at the ways of establishing the most important individuals for broadcasting and receiving of messages related to community bridging roles. We illustrated our findings with two examples of real-life networks, MIT reality mining data set of daily communications between 106 individuals during one year and UK Twitter mentions network, direct messages on Twitter between 12.4k individuals during one week.

**Keywords:** evolving networks, spectral radius, centrality ranking.

## 1 Introduction

Today's interconnected world with millions of users of mobile devices, computers and sensors leaving digital traces provides social scientists with previously unseen opportunities to create and validate their theories on a large scale. These social networks, captured in digital world, present us with research challenges: they are large, multi-layered and dynamic, i.e. they evolve from moment to moment. Thus, there is a need for the methods developed for regular and arbitrary static networks to be extended and adapted to dynamic, evolving networks.

One of the very important and well researched characteristics of an individual (a node) in a social network is its centrality score. Centrality measures the relative importance of a node and determines its involvement in a network. Although different centrality measures were proposed, tested and compared on undirected,

directed and weighted networks (for reviews see [3,16]), only relatively recently research focused on centrality in evolving networks [9,13].

For static networks, Katz centrality [15] computes the relative influence of a node within a network by measuring the number of the immediate neighbors and all the other nodes in the network that connect to the node under consideration through the immediate neighbors. Walks made to distant neighbors are penalised by an attenuation factor  $\alpha$ . This concept was recently revisited in [9,13]. Communicability across time-steps is based on the extension of Katz centrality to evolving networks. The concept is already successfully implemented on a small scale mobile phone and email communication networks [13,12] and C. Elegans brain networks [6], however scaling it up to very large data-sets involves handling large matrices.

Another measure of centrality in static networks, Bonacich centrality [1], [2] introduces another parameter, similar to Katz centrality, but penalising direct and indirect links. Recently, this measure was revisited in [11] where the authors investigated whether the measure converges, and proposed a normalised variant. Although their motivation came from the claim that computing a spectral radius  $\rho_A$  (the largest eigenvalue of an adjacency matrix of a graph) is difficult, “especially for large networks” (see pp. 2 of [11]), we note that in social network analysis settings where networks are mostly sparse, power iteration or similar methods could be used, that efficiently compute an approximated value of  $\rho$  even for very large sparse matrices, or else Perron-Frobenius theorem (see e.g. [10]) provides simple but useful bounds. However, we argue that a constraint such as  $\alpha < \frac{1}{\rho(A)}$  is limiting and should be relaxed for different reasons, particularly as it might penalise too heavily not-so-long paths, and thus lower significantly the centrality ranking of nodes that connect different communities which might have implications in large social network analysis. The “structural holes” theory [4] refers to the absence of links between two parts of a network. Brokerage exploits structural holes – an individual is connected to two other individuals or communities not mutually connected. This position could be beneficial for such an individual (a broker) as she/he could control a flow of information between two communities, profit from two different sources of information and mediate trade between them. Also bridges between different communities in a social network are important when trying to identify communities in the large unknown network and to run a network-based intervention which depends on community structure to change behaviour of individuals in the network [17].

In the empirical analysis of several real-world and artificial generic models of networks, Jamaković et al. [14] looked at the different upper bounds of spectral radius from the simple bound given by the graph’s maximal degree to more complex bounds featuring local information (average neighbours degree) or global information such as diameter. They found that for three real-world networks they investigated, a bound given by [7] was the closest to the observed values, while for the Internet autonomous systems topology the same bound was overestimating the real value significantly. When using artificial generic networks (random, small-world, preferential attachment networks) with the same number

of nodes and edges as in the real-world networks, the spectral radii of all three types of networks were much smaller than the real-world one in the Internet AS topology case. This is important because the spectral radius is also found to be connected to epidemic spreading in networks (see [18,5]).

In the following section we discuss how communicability indices are related to spectral radius and propose a new centrality measure which relaxes convergence constraints previously imposed by the spectral radius. We then create vitality measure based on centrality indices and show how to detect the individuals whose lack of existence would result in the biggest changes in centrality in evolving networks. We apply our findings to two real-life networks, and conclude with the discussion.

## 2 Relaxed Communicability

An evolving network is a family of graphs  $G_i = (V, E_i)$ , where the vertex set  $V$  is given in advance and is fixed throughout time, and an edge set  $E_i$  is a set of edges on  $V$  in the time  $i$ . We assume that the time is discrete and finite, i.e.  $i = 1, \dots, n$ . The corresponding adjacency matrices are denoted with  $A_i$ .

### 2.1 Communicability

For a static matrix  $A$ , the Estrada-Hatano communicability indices for a matrix  $A$  can be obtained from a communicability matrix as the row/column sums, given by

$$Q = e^A \quad (1)$$

where the matrix exponential of a matrix  $A$  is defined as

$$e^A = \sum_{k=0}^{\infty} \frac{1}{k!} A^k \quad (2)$$

This can be extended to evolving networks by  $Q = \prod_{i=0}^n e^{A_i}$ , which can be computed directly for small-scale networks. Another version of a communicability matrix, closer to the Katz definition, from [13] is given by:

$$Q = \prod_{i=0}^n (I - \alpha A_i)^{-1} \quad (3)$$

where  $I$  is identity matrix,  $\alpha < \frac{1}{\max(\rho(A_i))}$ ,  $i = 0, \dots, n$  denotes consecutive time-steps and  $\rho(A_i)$  is the spectral radius of  $A_i$ . Henceforth, we refer to (3) as the “*standard communicability*”. Broadcast and receive indices are equal to the row, resp. column sums of  $Q$ .

## 2.2 Spectral Radius Bound

In the case where existing communicability indices are used (3), Katz centrality for each  $A_i$  can be written as

$$(I - \alpha A_i)^{-1} = \sum_{k=0}^{\infty} \alpha^k A_i^k \quad (4)$$

and in order that (4) converges in standard matrix norm, one has that the attenuation factor  $\alpha < \frac{1}{\rho(A_i)}$  and similarly  $\alpha < \frac{1}{\max(\rho(A_i))}$ ,  $i = 0, \dots, M$  for (3). On the other hand, looking at each individual  $A_i$ , if  $\alpha$  is interpreted as a probability that, once sent, a message will be successfully transmitted by any receiving node to any of its contacts, then the expected length of a single transmission sent from nodes in the network corresponding to  $A_i$  is

$$\sum_{k=0}^{\infty} k \alpha^k (1 - \alpha) = \frac{\alpha}{(1 - \alpha)} \quad (5)$$

This implies for matrices with a spectral radius of more than 3, we must choose  $\alpha < \frac{1}{\max(\rho(A_i))}$ , the expected value of transmission length will be less than  $\frac{1}{2}$ , and if a spectral radius is greater than 2, the expected transmission length is less than 1.

As it was shown in [14] some of real-world networks have radius greater than 2, resulting in expected path lengths between 1 and 2, which means that especially paths between two communities are too heavily penalised. In order to mitigate the attenuation, we propose to normalise  $A$  and relax the condition on the attenuation which allows for longer paths and so rewards individuals that act as bridges between different communities appropriately.

## 2.3 Relaxed Communicability

For a large data-set where the size of the matrix  $Q$  is prohibitive, and computing inverse of such a large matrix represents a challenge, an approximation of  $Q$  can be computed using a Taylor series approximation ignoring summands of order higher than some  $n$ , depending on the application. In order to compute  $(I - \alpha A)^{-1} \mathbf{1}$  without storing  $Q$ , the following method can be used where  $\mathbf{b}$  is initialised to the all ones vector of length  $n$ .

$$(I - \alpha A)^{-1} \mathbf{b} = \mathbf{b} + \alpha A \mathbf{b} + \alpha^2 A^2 \mathbf{b} + \dots \quad (6)$$

We will use this representation to define new relaxed communicability indices. Instead of having  $\alpha = \frac{1}{2 \max(\rho(A_i))}$ , for the expression to converge, it is enough for  $\alpha$  to be less than 1, and that  $A$  is normalised. From the expression for the expected path length (5), ensuring that

$$\alpha < \frac{l}{1 + l} \quad (7)$$

where  $l \in \mathbb{N}$  is the expected path length, we have that  $\alpha$  will always be less than 1 and we can set parameter  $l$  on a desired path length depending on a context, i.e. what kind of centrality we are interested in. Thus, to obtain relaxed communicability indices, one should choose a length of path  $l$  depending on the application, calculate  $\alpha$  from (7) for given  $l$ , initialise  $b$  to be all-ones vector and multiply it with  $\alpha$  and matrix  $A_i$  normalised with 2-norm of  $A_i$  iteratively. Summing up all iterative factors up to the order  $n$ , which depends on how small one's approximation error needs to be, gives the result for  $A_i$ . Results need to be multiplied for all consecutive  $A_i$ s. In the case of a small graph,  $Q$  can be obtained directly from (3) using computed  $\alpha$  and replacing  $A$  with  $\frac{A}{\|A\|}$ .

## 2.4 Vitality Measure

In order to rank the nodes by importance during a time period we formulated vitality-based measure by computing the corresponding centrality indices in the absence of one node at time. For a series of adjacency matrices  $A_{i_1}, \dots, A_{i_2}$  we compute communicability indices using both standard and relaxed communicability indices. Furthermore, for each vertex  $k$ , we compute  $Q_k$ , which is obtained deleting exactly the  $k$ th row and the  $k$ th column from  $A_{i_1}, \dots, A_{i_2}$ , and then calculating both versions of communicability. Then we calculate the difference between  $Q_k$  indices and  $Q$  for each  $k$ , as a sum of least squares to check which nodes are responsible for the biggest changes in indices' values. We give a pseudo-code for vitality measure on Fig 1, which is independent of the version of communicability used (standard or relaxed). We will discuss in the next section how results depend on the type of communicability used.

## 3 Applications

We used two real-world data sets. The first one is the mutual mobile phone communications over a year for 106 individuals which was captured as a part of the MIT reality mining data set [8]. The second is the data-set obtained from Twitter UK mentions network collected on our behalf by Datasift, Twitters certified partner. The network was created from public messages that users located in UK sent to each other on Twitter using @ sign during 1 week in Dec 2011. In both cases we aggregated data on daily basis.

### 3.1 Case-Study 1: MIT Reality Mining Data

Data is aggregated on a daily basis, and contains 365 binary adjacency matrices, from the 20th July 2004 onwards, denoted with  $A_1$  to  $A_{365}$ . An entry  $(i,j)$  of  $A_k$  is equal to 1 if there was at least one phone call between  $i$  and  $j$  on day  $k$ . On Fig 2 given are the spectral radii of all 365 matrices. One can observe how the communication structure changes through the year.

On Fig 3 we show an example of a daily communication network. The vertices with labels 10, 45, 59 and 71 (highlighted on Fig 3) have relatively small degree,

## VITALITY MEASURE

```

compute indices (column and row sums of Q for A_i_1..A_i_2)
ls=0
for j=1:N
    remove A_k(j, .) and A_k(., j) for k=i_1,..,i_2
    compute indices_j (column and row sums of Q_j)
end
for i =1:j-1
    ls(j,1)=ls(j,1)+(indices_j(i,1)-indices(i,1))^2;
    ls(j,2)=ls(j,2)+(indices_j(i,2)-indices(i,2))^2;
end;
for i=(j+1):n
    ls(j,1)=ls(j,1)+(indices_j(i-1,1)-indices(i,1))^2;
    ls(j,2)=ls(j,2)+(indices_j(i-1,2)-indices(i,2))^2;
end;
ls=sqrt(ls)

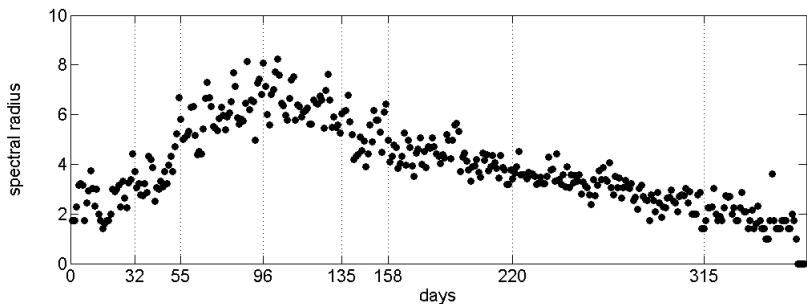
```

**Fig. 1.** Computing vitality measure. Indices are  $2 \times n$  array - the first column is  $Q$ 's column sum, and the second is  $Q$ 's row sum.  $Q_j$  is obtained from all  $A_i$ 's removing  $j$ -th column and  $j$ -th row from each adjacency matrix, and  $\text{indices}_j$  are then column and row sum of  $Q_j$ .

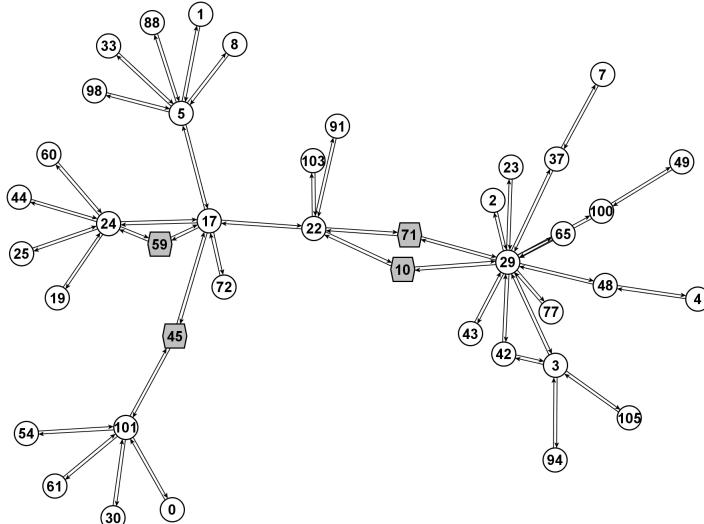
but they connect different communities and therefore are important. We picked a sample of seven daily networks on the days 32, 55, 96, 135, 158, 220 and 315, looking at the different values of spectral radii. We computed communicability indices using the standard and relaxed versions. The Table 1 presents results of rankings in descending order (1 top to 106 bottom) in both cases, showing much higher rankings when the relaxed version with the length of path 3 was used. On Fig 4 we show scatter plot of standard vs. relaxed broadcast indices (left panel) and standard vs. Estrada-Hatano communicability indices on the right panel. The upper left diagonal of the figure represent nodes that have higher rankings in relaxed than in standard indices. Note that in both standard and relaxed indices more weight for broadcast indices lies on the first matrix in the sequence, while for receive indices it is the last matrix that carries most of weight. While Estrada-Hatano indices do not correlate with standard or relaxed indices, they still rank higher most of community bridges. Thus Estrada-Hatano indices could be used when the expected transmission length is not known, but if the length of transmission is important, our parametric approach will highlight more relevant nodes.

### 3.2 Case-Study 2: Twitter Mentions Network Data-Set

The data-set comprised of around a million of tweets between UK users that contained mention of another UK user (sign @). The nodes represented the



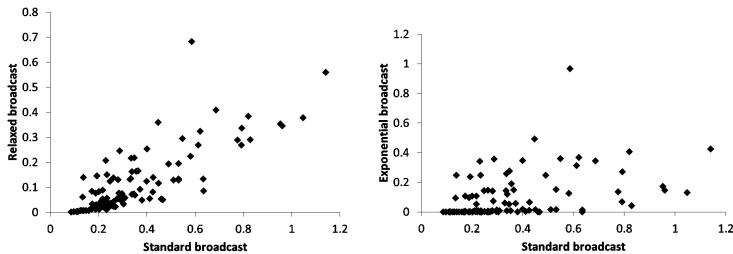
**Fig. 2.** MIT data:Spectral radii of  $A_1$  to  $A_{365}$  matrices



**Fig. 3.** MIT data: An example of a daily network (its largest connected component), on the day 32

**Table 1.** Ranking (in descending order, top 1 to bottom 106) of broadcast vs. relaxed broadcast

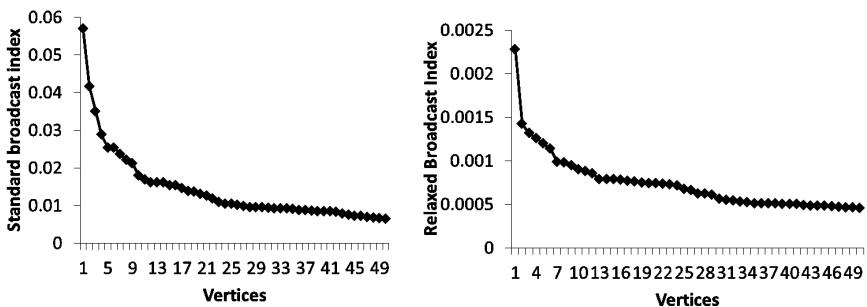
| Vertex | Rank (broadcast) | Rank (relaxed broadcast) |
|--------|------------------|--------------------------|
| 10     | 71               | 21                       |
| 45     | 35               | 25                       |
| 59     | 85               | 28                       |
| 71     | 71               | 21                       |



**Fig. 4.** Standard vs. relaxed broadcast indices, left, and standard vs. exponential broadcast indices, right

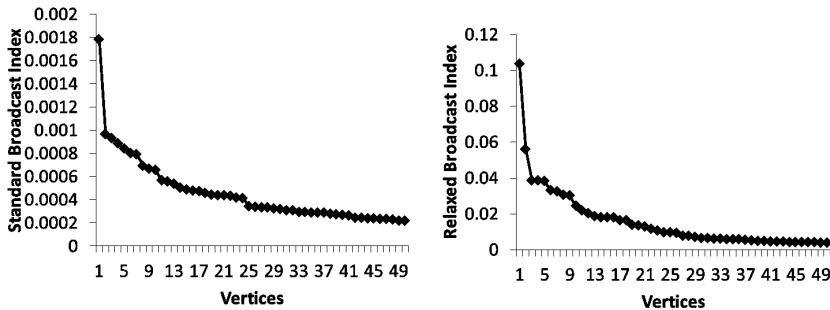
users, and if user A's tweet contained "@B", an edge between A and B was created. Only reciprocated edges were kept and multi-edges were ignored. All daily tweets were aggregated into a daily network, so we finished with 7 daily undirected graphs with 12408 nodes and around 2.7k edges in average. We computed both communicability and relaxed communicability indices, both using rank obtained from communicability, and rank obtained from vitality based measure (deleting each node and computing the sum of differences for all the other nodes as described earlier).

### 3.3 Results



**Fig. 5.** Top 50 vertices according to the ranking based on standard (left) and relaxed (right) broadcast

Although the computation of vitality measure is quite demanding (one needs to recompute communicability matrices for each node once) this is feasible as the daily networks are quite sparse. At 12408 vertices and 7 time-steps, this collection contains relatively big, but not large networks. Their broadcast indices decrease quickly so we ranked the indices from largest to smallest with respect



**Fig. 6.** Top 50 vertices according to the ranking based on standard (left) and relaxed (right) vitality

to broadcast and looked into more details at the first fifty indices. On Fig 6 one can see the difference in ranking between the two methods. Several vertices that are ranked much higher in relaxed than in standard broadcast index correspond again to vertices with relatively small degrees and were picked up as they connect different communities (e.g. vertex ranked 39 in relaxed is ranked 278 in normalised and has a degrees equal to  $(2, 0, 0, 3, 1, 0, 0)$  respectively in 7 daily networks.)

## 4 Conclusions

We used communicability indices to rank the nodes in evolving communication networks. While the computation of communicability for small-data sets is relatively simple and fast, for the large data-sets it means handling of large matrices, so one can use a Taylor approximation. We introduced a parameter (transmission length) that allows for targeting specifically brokers or bridges between communities. We have applied this approach on two real-life evolving networks obtained from mobile phone communications and Twitter. Using the vitality based measure, we proposed a way to rank vertices depending on the amount of change their communication abstinence would bring to the rest of the evolving network. We hope that a parametric approach that can be optimised according to a particular application will be a useful addition to a standard evolving social network analysis toolbox, especially when the expected length of message/communication transmission plays an important role, i.e. it is either given or can be approximated.

**Acknowledgments.** This work is funded by the RCUK Digital Economy programme via EPSRC grant EP/G065802/1 'The Horizon Hub' and EPSRC MOLTEN EP/I016031/1. We would like to thank Datasift for providing us with the Twitter dataset.

## References

1. Bonacich, P.: Power and centrality: A family of measures. *American Journal of Sociology* 92, 1170–1182 (1987)
2. Bonacich, P., Lloyd, P.: Eigenvector-like measures of centrality for asymmetric relations. *Social Networks* 23(3), 191–201 (2001)
3. Borgatti, S.P., Everett, M.G.: A graph-theoretic perspective on centrality. *Social Networks* 28(4), 466–484 (2006)
4. Burt, R.S.: Brokerage and closure: An introduction to social capital. *Eur. Sociol. Rev.* 23(5), 666–667 (2007)
5. Castellano, C., Pastor-Satorras, R.: Thresholds for epidemic spreading in networks. *Physical Review Letters* 105, 218701 (2010)
6. Crofts, J.J., Higham, D.J.: Googling the brain: Discovering hierarchical and asymmetric network structures, with applications in neuroscience. *Internet Mathematics* (Special Issue on Biological Networks) (2011)
7. Das, K.C., Kumar, P.: Some new bounds on the spectral radius of graphs. *Discrete Mathematics* 281(1-3), 149–161 (2004)
8. Eagle, N., Pentland, A.S., Lazer, D.: Inferring friendship network structure by using mobile phone data. *Proceedings of the National Academy of Sciences* 106, 15274–15278 (2009)
9. Estrada, E., Hatano, N.: Communicability in complex networks. *Physical Review E* 77 (2008)
10. Gantmacher, F.: *The Theory of Matrices*, vol. 2. AMS Chelsea Publishing (2000)
11. Ghosh, R., Lerman, K.: Parameterized centrality metric for network analysis. *Physical Review E* 83(6), 066118+ (2011)
12. Grindrod, P., Higham, D.J.: Models for evolving networks: with applications in telecommunication and online activities. *IMA Journal of Management Mathematics* (2011)
13. Grindrod, P., Higham, D.J., Parsons, M.C., Estrada, E.: Communicability across evolving networks. *Physical Review E* 83 (2011)
14. Jamaković, A., Kooij, R.E., Van Mieghem, P., van Dam, E.R.: Robustness of networks against viruses: the role of the spectral radius. In: *Symposium on Communications and Vehicular Technology*, pp. 35–38 (November 2006)
15. Katz, L.: A new index derived from sociometric data analysis. *Psychometrika* 18, 39–43 (1953)
16. Opsahl, T., Agneessens, F., Skvoretz, J.: Node centrality in weighted networks: Generalizing degree and shortest paths. *Social Networks* 32(3), 245 (2010)
17. Valente, T.: Network interventions. *Science* 337(6090) (2012)
18. Wang, Y., Chakrabarti, D., Wang, C., Faloutsos, C.: Epidemic spreading in real networks: An eigenvalue viewpoint. In: *SRDS*, pp. 25–34 (2003)

# Finding Network Communities Using Random Walkers with Improved Accuracy

You Li<sup>1</sup>, Jie Wang<sup>1</sup>, Benyuan Liu<sup>1</sup>, and Qilian Liang<sup>2</sup>

<sup>1</sup> Department of Computer Science, University of Massachusetts, Lowell, MA 01854

<sup>2</sup> Department of Electrical Engineering, University of Texas at Arlington, TX 76019

**Abstract.** Finding communities in structural networks (online social networks included) with sufficient accuracy is an important issue. We present a new method to identify communities that are in the same order of time complexity as the existing algorithms. In particular, we present an efficient algorithm using random walkers which, on a given network, generates a new network to better reveal the structures of the original network. We then use existing hierarchical clustering algorithms on the new network to find communities. We carry out simulations on both computer-generated data and the widely-used karate club data [10], and show that our algorithm can identify communities with much improved accuracy.

**Keywords:** Accuracy of Community Identification, Random Walkers.

## 1 Introduction

Given a structural network represented as a graph, weighted or unweighted, directional or nondirectional, network communities are clusters of nodes within which the connecting edges are dense, while the connecting edges between communities are sparse. We assume that each node may belong to only one community.

Early algorithms for finding communities can be divided roughly into two categories: graph partitioning [9] and hierarchical clustering [6]. There are two challenges in devising community identification algorithms. One is the accuracy; we want to identify communities with sufficient accuracy. The other is the efficiency; we want to identify communities efficiently, hopefully in linear time, which is much more desirable for very large networks consisting of millions of nodes such as online social networks such as Facebook and Twitter. This paper is focused on the issue of improving accuracy of existing algorithms, while keeping the time complexity at the same level of the existing algorithms. Our idea is to first convert a given network into a new network that can better reveal the structure of the original network. We then apply an existing algorithm on the new network to identify communities. Thus, we must ensure that the conversion can be done efficiently, and should now incur higher time complexity than the existing algorithms. We devise an efficient algorithm using random walkers to achieve this goal. This algorithm, called Random Walker Conversion (RWC), can be run in quasi-linear time (or even linear time) for small-world networks.

We carry out simulations on both computer-generated data and the widely-used karate club data [10], and show that our algorithm can identify communities with much improved accuracy, which can reach 100% accuracy for these datasets with an appropriate number of iterations for RWC.

The paper is organized as follows. We provide a brief review in Section 2 on early approaches for finding communities. We then describe in Section 3 our RWC algorithm and provide theoretical analysis. In Section 4 we describe our simulations and present simulation results. We conclude the paper in Section 5.

## 2 Review of Existing Methods

We first describe an early algorithm based on graph partitioning, proposed by Kernighan and Lin [7]. We then describe two widely accepted algorithms based on hierarchical clustering.

### 2.1 Graph Partitioning

On a given (weighted) graph  $G = (V, E)$  and a positive integer  $k$ , the graph partitioning problem divides  $G$  into  $k$  components such that the size difference of any two components is at most 1 and the number of edges (or the summation of edge weights) between different components is minimized. This problem has important applications in task scheduling under multi-processor systems, sparse matrix reordering, parallelization of neural net simulations, particle calculation, and VLSI design, to name just a few, and it is NP-hard. The Kernighan-lin algorithm is a heuristic algorithm widely in VLSI design, and it be used to obtain rough network communities.

### 2.2 The Kernighan-Lin Algorithm

Let  $G = (V, E)$  be a weighted graph with weight function  $w : E \rightarrow R$  of real values. Let  $C_1$  and  $C_2$  be two different communities. Define the weight between these communities, denoted by  $\mathcal{P}_{C_1, C_2}$ , as the summation of the weights of all edges between  $C_1$ ,  $C_2$ . That is,

$$\mathcal{P}_{C_1, C_2} = \sum_{u \in C_1, v \in C_2} w(u, v). \quad (1)$$

The Kernighan-Lin algorithm divides the graph  $G$  into two subgraphs  $C_1$  and  $C_2$  such that the size difference is at most one, and in a sequence of iterations swaps a node  $u \in C_1$  with a node  $v \in C_2$  to form two new communities  $C'_1$  and  $C'_2$  such that  $\mathcal{P}_{C_1, C_2} - \mathcal{P}_{C'_1, C'_2}$  is maximized. This algorithm incurs a time complexity of  $O(|V|^3)$ .

### 2.3 Hierarchical Clustering

Hierarchical clustering is a widely used method in finding communities in a network. The basic idea of this method is to find relations among nodes in the network, and then extract communities from the network by either adding edges or deleting edges. Hierarchical clustering can be represented in a tree data structure known as dendrogram. It can be further divided into divisive clustering and agglomerative clustering.

**Divisive Clustering.** Divisive clustering is a top-down approach. It deletes a critical connected edge and performs splits recursively as the algorithm moves down the hierarchy. The most popular algorithm of this kind was devised by Girvan and Newman [8], who defined the “edge betweenness” of an edge as the number of shortest paths between pairs of nodes that run along it. Thus, the edges connecting communities will have higher edge betweenness (at least one of them). By removing these edges, the groups are separated to reveal the underlying community structure. In this process a dendrogram is created from the root to the leaves. Each time an edge is removed, the algorithm has to recalculate the betweenness for all the remaining edges.

The time complexity of Girvan and Newman algorithm is high. For each iteration, computing the betweenness of all existing edges in the network incurs a time complexity of  $O(|V||E|)$ . A total of  $|E|$  edges will finally be removed. Hence, the worst-case running time is  $O(|V||E|^2)$ .

**Agglomerative Clustering.** Agglomerative clustering is a bottom-up approach. It uses a weight function on edges to determine how the network should be divided. Weight functions are either based on the metrics of similarity or the strength of connection among nodes. A widely used weight function for an edge depends on the neighbors of the two nodes connected by the edge: If the nodes  $i$  and  $j$  have similar sets of neighbors, then they are considered structurally similar and so they belong to the same cluster [8]. An example of such a similarity measure is given below [4]:

$$\mathcal{S}_{i,j} = \sqrt{\sum_{k \neq i,j} (A_{i,k} - A_{j,k})^2} \quad (2)$$

where  $A_{ij}$  is the element of adjacency matrix of the graph corresponding to the nodes  $i$  and  $j$ . We can see that when  $(\mathcal{S})_{i,j} = 0$ , nodes  $i$  and  $j$  have exactly the same set of neighbors.

Agglomerative clustering starts with an empty network. It runs in a number of iterations. In each iteration, an edge is added based on the weight function. As more edges are added, connected subsets begin to form, which represent communities. In this process a dendrogram is created from the leaves to the root. Running the agglomerative method requires a pre-determination of how large each community should be. This is a drawback for finding natural communities.

Also, the structural similarity is limited since the nodes in the same community may not necessary share even one neighbor. In general, agglomerative clustering is faster than divisive clustering, which incurs a complexity of  $O(|V|^2)$ .

### 3 Randomized Algorithm with Random Walkers

Fang and Wang [5] showed how to use random walkers to effectively identify structural similarities between different networks in linear time. We observe that using random walkers we may also convert a given network into a new network that better reveals the structures of the original network, and we may do so in quasi-linear time or even in linear time.

The basic idea is as follows: We generate for each node in the network a number of walkers and let them walk at random from one node to another. We have the following two observations: (1) A random walker is highly likely to stay in the same community in a random walk. That is, if a walker starts from node  $i$  and ends at node  $j$ , then node  $i$  and node  $j$  would have a high probability to be in the same community. (2) The sets of walkers on any two nodes in the same community are likely to have certain similarities. For instance, in a few random walk steps, node  $i$  and node  $j$  in the same community may both have walkers from node  $k$ .

Our algorithm, called *Random Walker Conversion* (RWC), consists of three phases. In the first phase, RWC generates an appropriate set of walkers on each node and label the walkers with the node index. The exact number of walkers to be generated will be determined later. In the second phase, RWC proceeds in a predefined number of iterations. In each iteration, each walker chooses a destination node following a defined set of rules and moves to the node. In the third phase, RWC computes the corresponding sets of walkers for any two nodes and generates a new network, which better reveals the structures of the original network.

#### 3.1 Phase I and Phase II

We refer to the node for which the walker is generated as the *owner node* of the walker and the node on which the walker is currently on as the *current node* of the walker. In Phase II we define a set of rules for walkers to choose destination nodes in each iteration.

Each walker chooses from two possible actions. With probability  $p$ , the walker moves back to its owner node. With probability  $1 - p$ , the walker selects a neighbor node of the current node as the destination and moves to it. In this case, the neighbor node is selected based on the weight of the connecting edge. The walker must select a node that is different from the one it travels from in the previous iteration. If the current node has only one neighbor, i.e., the current node is a leaf node, then the walker will never move again; namely, it stays on the current node until the end of all iterations. Such a walker is referred to as a *stationary* walker. Let  $w_{i,j}$  denote the edge weight between node  $i$  and node  $j$ .

Let  $\mathcal{R}_i$  denote the neighbor set of node  $i$ . The probability  $\gamma_{ij}$  that a walker on node  $i$  chooses neighbor node  $j$  is defined by

$$\gamma_{ij} = \frac{w_{ij}}{\sum_{k \in \mathcal{R}_i} w_{ik} - w_{iv}}, j \neq v, \quad (3)$$

where  $v$  is the node where the walker came from in the previous iteration.

**Proposition 1.** *Given a tree with depth  $H$ , if we generate  $\mathcal{S}$  walkers on root node  $i$  and let them each walk  $t$  steps, then the expected number  $n_h$  of walkers on the nodes at level  $h$  at the end of phase II is given by*

1. If  $t < H$ , then

$$n_h = \begin{cases} \mathcal{S}p(1-p)^h, & \text{if } h < t, \\ \mathcal{S}(1-p)^h, & \text{if } h = t. \end{cases}$$

2. If  $t \geq H$ , then

$$n_h = \begin{cases} \mathcal{S}(1 - (1-p)^H)p(1-p)^h, & \text{if } h < H, \\ \mathcal{S}(1-p)^H, & \text{if } h = H. \end{cases}$$

*Proof.* We compute the probability a walker arrives at each level of the tree.

1. If  $t < H$ , then it is not possible for the walker to walk to a leaf node. If  $h < t$ , no matter how walkers move in previous iterations, in the last  $h+1$  iterations, it must have moved back to its owner node and then walk  $h$  iterations to a node at level  $h$ . The probability for this to happen is equal to  $p(1-p)^h$ . If  $h = t$ , it means that the walker never moves back and reaches level  $h$ . The probability for this to happen is equal to  $(1-p)^h$ .
2. If  $t \geq H$  and the walker reaches the deepest level  $H$ , then the walker will be trapped there until the end. The probability for this to happen is equal to  $(1-p)^H$ . If  $h < H$ , then the walker must have not reached level  $H$ . The probability for this to happen is equal to  $(1 - (1-p)^H)p(1-p)^h$ .

This completes the proof.

We will let walkers walk one step in each iteration. Proposition 1 indicates that on node  $j$ , the expected number of walkers coming from node  $i$  at the end of  $t$  steps decreases according to the distance from node  $j$  to node  $i$ . The analysis does not directly apply to graphs with circles. However, since circles only prevent walkers from traveling farther, the argument still holds; that is, the farther the node is from node  $i$ , the lesser expected number of the walkers from node  $i$  it will have at the end of the iterations. Thus, this argument conforms to the intuition that the farther the node is from node  $i$ , the lesser possible it is in the same community as node  $i$ .

We now determine how many walkers should be generated on each node in Phase I. As for node  $i$ , the nodes that are most likely in the same community as  $i$  are node  $i$ 's neighbor nodes. Thus, we generate walkers on node  $i$  such that after the iterations, the neighbor nodes of  $i$  are expected to have at least one walker

from  $i$ . Also, the edge weight is converted to the expected number of walkers. It follows from Proposition 1 that it is sufficient to generate the following number of walkers on node  $i$ :

$$\mathcal{S}_i = \frac{\sum_{k \in \mathcal{R}_i} w_{ik}}{p(1-p)}. \quad (4)$$

Together with the probability discussed in Section 3, we note that if node  $i$  and  $j$  are neighbor nodes, then it is expected that node  $j$  will have  $w_{ij}$  walkers from node  $i$  at the end of random walk iterations.

Details of Phase I and Phase II are given in Algorithm 1.

---

**Algorithm 1.** Phase I and Phase II of RWC on an unweighted network

---

```

1: Let  $d_v$  denote the degree of node  $v$  in the graph
2: for all node  $v$  in  $V$  do
3:   generate  $d_v/p_u$  walkers
4:   set owner node of walkers as  $v$ 
5: end for
6: for all generated walker  $w$  that is not stationary do
7:   with probability  $p_u$ ,  $w$  moves back to its owner node, set  $w$ 's current node to
      the owner node, and set its previous node null.
8:   if  $w$  does not move back to its owner node then
9:     if current node of  $w$  has degree 1 then
10:    set  $w$  to stationary
11:   else
12:     randomly choose a neighbor node  $y$  of  $w$ 's current node  $x$  following
        Equation (3)
13:     walk to  $y$ , set  $w$ 's current node to  $y$ , and  $w$ 's previous node to  $x$ .
14:   end if
15:   end if
16: end for
```

---

### 3.2 Phase III

After the iterations of random walks are finished, each node will have a certain number of walkers on it. We generate a directed weighted network as follows: First, remove all existing edges from the original network. Second, add an directed edge from node  $i$  to node  $j$  if node  $j$  has  $w_{ij}$  neighbors from node  $i$ , and assign a weight  $w'_{ij}$  to this edge defined by

$$w'_{ij} = \frac{w_{ij}}{W_i}, \quad (5)$$

where  $W_i = \sum_{k \in \mathcal{R}_i} w_{ik}$  is the sum of weights of all connecting edges to node  $i$  in the original network. Note that normalization is used to help better reveal the structures of the original network. Intuitively, the more walkers node  $j$  has from

node  $i$ , the more possible the two nodes are in the same community. However, if node  $i$  generates only two walkers and they move to node  $k$  while node  $j$  generates 100 walkers and two of them move to node  $k$ . Without normalization node  $i$  and node  $j$  may have been wrongly considered similar.

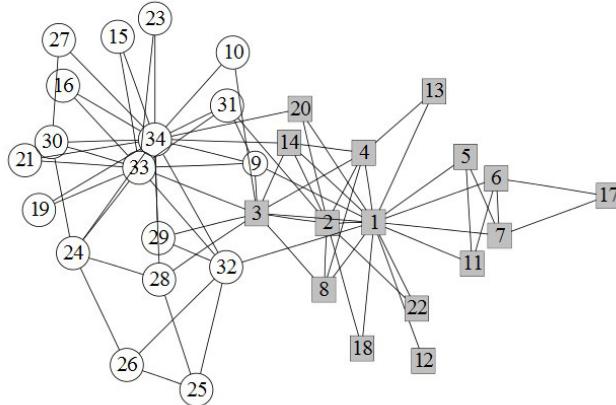
The time complexity of RWC is  $O(t(|V|d))$ , where  $t$  is the number of iterations and  $d$  an average node degree. For structural networks such as social networks (online or offline) that are small world [1,3,2], the average length between any two nodes is proportional to  $\log |V| / \log d$ , and so we may choose  $t < \log |V|$  or even a constant. Moreover, in small-world networks it is common that  $d$  is much smaller than  $|V|$ . Thus, the time complexity of RWC is close to quasi-linear for such networks or even linear time if we set  $t$  to be a constant.

## 4 Simulations

We evaluate the accuracy of finding network communities using RWC and existing clustering algorithms via numerical experiments. We consider two scenarios. In the first scenario, we generate at random a number of unweighted networks with 1,000 nodes, where in each graph, 5 communities of equal size are formed so that we know the true communities. In particular, the nodes in the same community are connected with probability 0.75 while the nodes in the different communities are connected with probability 0.1. In the second scenario, we used the widely-used “karate club” network [7], which is a weighted social network with nodes representing members of a karate club observed by Zachary for roughly two years in the 1970’s, and edges indicating social interaction between the connecting members. During the two year period, the administer and the coach of the club had a fight, and the coach finally left the club along with his customers. In this network, there are 34 nodes and after the departure of the coach, the network is split into two communities. One consists of 16 nodes:  $\{1,2,3,4,5,6,7,8,11,12,13,14,17,18,20,22\}$ , and the other consists of 18 nodes:  $\{8,9,14,15,18,20,22,23,24,25,26,27,28,29,30,31,32,33\}$ . The figure 1 [6] shows the structure of the community with the core node 1 and 34 represent the coach and administer respectively.

In each scenario, we run agglomerative clustering on the original network and the networks generated by RWC. The clustering algorithm runs in iterations. Initially, each node itself forms a cluster. In each iteration, two clusters with the most similarity are merged. We use the Pearson correlation coefficient to evaluate similarity between any two nodes [8]). The Pearson correlation of node  $i$  and  $j$ , denoted by  $x(i, j)$ , is given by

$$x_{i,j} = \frac{\frac{1}{n} \sum_k (e_{i,k} - \mu_i)(e_{j,k} - \mu_j)}{\sigma_i \sigma_j}, \quad (6)$$



**Fig. 1.** The Karate Club Community Graph

where

$$\mu_i = \frac{1}{|V|} \sum_k w_{i,k},$$

$$\sigma_i = \frac{1}{|V|} \sum_k (w_{i,k} - \mu_i)^2$$

are the mean and variance of node  $i$ . When the value of  $x_{i,j}$  is larger, nodes  $i$  and  $j$  are more likely in the same community. The similarity between two clusters is computed using the mean similarity between elements of each cluster. In particular, the similarity between two cluster  $C_1$  and  $C_2$  is computed by

$$\frac{1}{|C_1||C_2|} \sum_{i \in C_1} \sum_{j \in C_2} x(i, j),$$

where  $|C|$  denotes the size of cluster  $C$ . The process continues until the number of remaining clusters is equal to a predefined threshold  $U$ .

#### 4.1 Scenario I

In this scenario, we run 100 simulations on 100 randomly generated networks consisting of 1,000 nodes. For each network we generate a new network using RWC with  $t = 5$  which is much smaller than  $\log 1,000$ . The average accuracy of finding the communities over 100 runs is computed as follows:

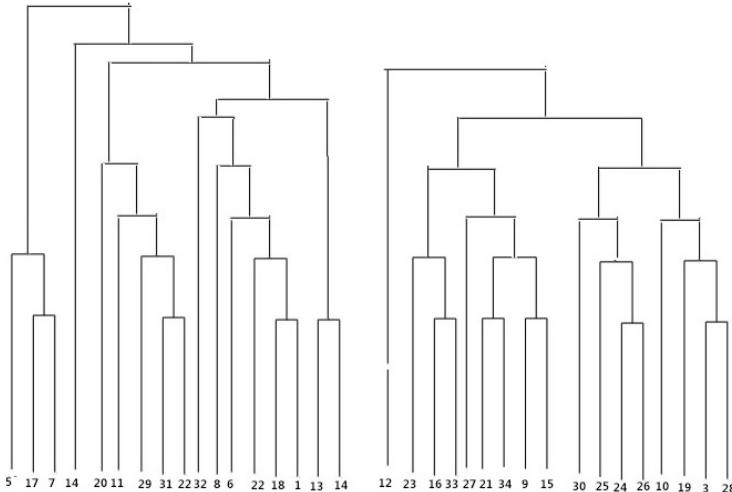
$$\text{Avg. Accuracy} = \frac{1}{N_r} \sum_{i=1}^{N_r} \sum_{i=1}^{N_c} \frac{f^i}{s^i},$$

where  $N_r$  is the total number of simulation runs, i.e.,  $N_r = 100$ ,  $N_c$  the number of communities, i.e.,  $N_c = 5$ ,  $f^i$  the number of members that were correctly found for community  $i$ , and  $s^i$  the size of the community  $i$ .

We find that using RWC greatly improves the accuracy. In all simulation runs, the agglomerative clustering on the network generated by RWC results in 100% accurate community division while the average accuracy of the agglomerative clustering on the original network is about 92%.

## 4.2 Scenario II

In this scenario, we let walkers walk at random with different random seed values. The agglomerative clustering on the original network results in the following community division:  $\{12\}$  and  $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34\}$ , which bear no correlation with the true communities. We run RWC 100 times with  $t = 3 < \log 34$  and generate 100 different networks. We found that agglomerative clustering on these RWC networks returns the same division:  $\{1, 2, 4, 5, 6, 7, 8, 10, 11, 13, 14, 17, 18, 20, 31\}$  and  $\{3, 9, 12, 15, 16, 19, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 32, 33, 34\}$ , which is about 70% accurate compared to the true communities (see Figure 2).



**Fig. 2.** The hierarchical tree generated by the random walker method

We note that in both scenarios, using RWC to generate new networks only incurred a very small amount of extra time, which was negligible compared to the running time of the community identification algorithms.

## 5 Conclusion

We presented a new algorithm using random walkers to improve the accuracy of network community identifications. Our algorithm first generates a new network from the original network using random walkers, which can better reveal the structures of the original network. It then uses an existing clustering algorithm on the new network to find communities. The time complexity of our algorithm is at the same level of the existing clustering algorithm. We show that, using numerical simulations, our algorithm can greatly improve the accuracy of community identifications over the existing clustering algorithms.

**Acknowledgement.** Y. Li and J. Wang were supported in part by the NSF under grant CNS-1018422. J. Wang was also supported in part by the NSF under grant CNS-1247875. B. Liu was supported in part by the NSF under grant CNS-1018303 and grant CNS-0953620. Q. Liang was supported in part by the NSF under grant CNS-1247848. The authors thank Zheng Fang and Weibo Gong for constructive discussions in the early stage of this work.

## References

1. —. Six degrees of separation's theory tested on facebook. *Telegraph* (August 17, 2011)
2. —. Six degrees of separation, Twitter style. *Sysomos* (April 30, 2010)
3. Barnett, E.: Facebook cuts six degrees of separation to four. *Telegraph* (November 22, 2011)
4. Burt, R.S.: Positions in networks. *Social Forces* 55(1), 93–122 (1976)
5. Fang, Z., Wang, J.: Efficient identifications of structural similarities for graphs. *Journal of Combinatorial Optimization* (May 2012), <http://link.springer.com>
6. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. *Proceedings of the National Academy of Sciences* 99(12), 7821–7826 (2002)
7. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal* (1970)
8. Newman, M.E.J.: Detecting community structure in networks. *The European Physical Journal B-Condensed Matter and Complex Systems* 38(2), 321–330 (2004)
9. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. *Physical Review E* 69(2), 026113 (2004)
10. Zachary, W.W.: An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 452–473 (1977)

# Homophilies and Communities Detection among a Subset of Blogfa Persian Weblogs: Computer and Internet Category

Adib Rastegarnia<sup>1</sup>, Meysam Mohajer<sup>1</sup>, and Vahid Solouk<sup>2</sup>

<sup>1</sup> University of Tehran

Dept of Information Technology Engineering

[adib.rastegarnia@ieee.org](mailto:adib.rastegarnia@ieee.org),

[meysammohajer@ieee.org](mailto:meysammohajer@ieee.org)

<sup>2</sup> Urmia University of Technology

Dept of Information Technology Engineering

[vsolouk@ieee.org](mailto:vsolouk@ieee.org)

**Abstract.** The investigation of relationships between various social actors has been the main focus of social network analysis in explaining the structure of social relations, measuring the relationships between the actors and etc. Blogfa is among the popular web service providers for building Persian weblogs in Iran. To the best of our knowledge, there is no social network analysis for the Computer and Internet Category of Blogfa Persian weblogs. The current paper presents a social network analysis for the Computer and Internet category of Blogfa. Each weblog in the target category contains a list of friends to which, it establishes a connection called links or links of friends. These links lead to the formation of a relationship network between weblogs. The current study has particularly focused on the relationship analysis of the network of weblogs based on the friend links. We report on our analyses and measurements of different centrality parameters such as in-degree, out degree, clustering coefficient, modularity for the group of weblogs. Furthermore, the degree of collaborations between these weblogs are analyzed and some homophilies are detected among them. It was found through the analyses that the majority of the bloggers tend to link the weblogs which provide the contents with subjects of common interests among the bloggers, and that the common interests are merely general subjects rather than professional ones.

**Keywords:** Social network analysis, Blogfa Persian weblogs, Clustering coefficient, Modularity, Relationships Network, Community Detection, Homophily detection.

## 1 Introduction

The term Social Network (SN) is used for a social structure that provides maps of dyadic ties between individuals and organizations such as friendship, kinship, common interest, financial exchange, etc [1,2]. SNs can operate in different levels

from the families up to the nations and thereby, play critical roles in determining the way problems are solved, organizations run, markets evolve and the degree in which individuals succeed in achieving their goals [3]. Hence, the analyses of SN with focus on studying of relationships between various social actors has been used by researchers to explain the structure of social relations, measure the relationships between the actors, etc [4,5]. It is also believed that SN analysis can be used to evaluate the performance of individuals, groups or the entire the social network.

Blogfa is one of the popular service providers for building Persian weblogs in Iran. It contains above 300,000 Persian Weblogs in different categories such as computer, sport, culture, business, entertainment, personal and etc [6].

Several studies on analyzing relationships in SN have been conducted on weblogs as briefly reported in Section 2. To the best of our knowledge, there is no SN analysis for the Computer and Internet Category of the Blogfa Persian weblogs. In this paper, a social network analysis for the Computer and Internet category of Blogfa is presented. Each weblog includes a list of friends to establish connection, called links or links of friends. These links play role in establishing a network of relationships among weblogs. The current study has mainly focused on the network of relationships of weblogs based on their friends' links. We have analyzed different centrality parameters such as in-degree, out degree, clustering coefficient, modularity for the group of the weblogs under investigation. Furthermore, the degree of collaborations between these webglos are analyzed and number of hemophilies are detected among them. The rest of the paper is organized as follows: the following section provides a brief overview of the related works on SN analysis of real world scenarios. Data gathering process is described in section 3. Section 4 presents some of the basic social network analyses based on centrality parameters such as degree distribution, clustering coefficient, connected components, modularity and K-core. The relationships between the number of incoming friends' links and content of the weblogs are explained in Section 5. Section 6 presents an analysis of global relationship network. Visualization and Analysis of the relationships between the Computer and Internet Category Weblogs are presented in Section 7, and Section 8 concludes the work.

## 2 Related Works

Social network analysis of real world scenarios has been the concern of several studies. In [5] an analysis of top50 political weblogs in people's daily web, based on centrality has been presented. Data mining and centrality analysis have been used to find the network links between the top50 political weblogs. In addition, in order to achieve better understanding of political blogs community, the authors have been tried to find some of the main political blogs groups such as core group members, members with special characteristics, and the important group members. A social network analysis has been presented in [7] for FIT community server (FITCS) which is a popular way for communication between FIT students. Some of the main social network characteristics such as density, closeness, degree

and betweenness have been measured for the mentioned social network. In addition, the analysis shows that FITCS can be considered as a small-world scale free network, with several hubs. Furthermore, a large scale study on Persian weblogs has been presented in [8]. Commenting behaviors of Persian bloggers are investigated and a simple model for distribution of comments is introduced by the authors. Social network analysis and data mining methods was used in [9] to investigate the network relationships in on-line forum of university. To solve some of the most important problems such as improving the communication in the university on-line forum, make the students more positive and learn more knowledge, some advices are proposed. In addition, in [10] structural features of the Sina's VIP Blogsphere and the behavior patterns of its members have been investigated by using social network analysis. The authors concluded the behavior patterns of Sinas VIP Blogsphere is consistent with real life behaviors of bloggers.

### 3 Data Collection Process

In this paper, two kinds of relationships between the weblogs are defined which are described as the follows:

- Local Relationships Network (LRN): denotes the existing links of friends among the Computer and Internet category of Blogfa Persian weblogs.
- Global Relationships Network (GRN): denotes the existing links among the Computer and Internet category and the other categories of Blogfa Persian weblogs such as Entertainment, sports, business, religious, etc.

We have used Win Web Crawler [11] in order to collect the list of weblogs and to extract their in-between relationships. In this study, a number of 16429 webblogs from the Computer and Internet category are crawled and a list of 64000 webblogs from other categories are extracted and categorized.

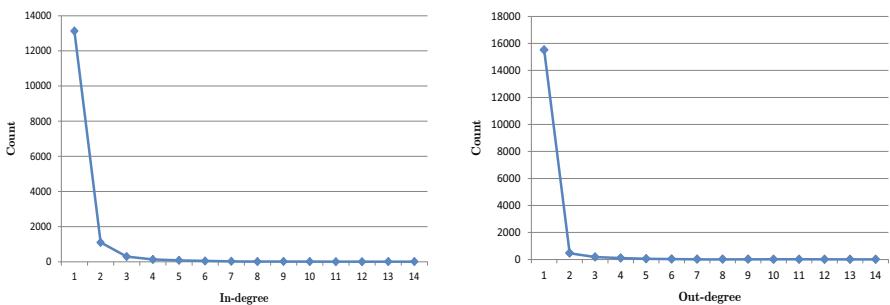
### 4 Basic Social Network Analysis

In this section, some basic analysis on the basis of degree, modularity, clustering coefficient and connected components parameters are examined on the LRN of weblogs. The Gephi software is used to visualize and analyze the relationships network of weblogs. Gephi platform is an open-source interactive visualization and exploration tool for all kinds of networks and complex systems [12].

#### 4.1 Degree Analysis

The degree of a node (a node refer to the each of the weblogs) denotes the number of links to a node. In the directional graph like the case in the current study, in-degree and out-degree has been applied to the number of links pointing in and out of a node, respectively [10]. In-degree and out-degree distributions of

LRN of weblogs are illustrated in Figure 1. As shown in Figure 1, only few of nodes have numerous incoming links when compared with the rest with only few incoming links. The nodes with many incoming links act as hubs or connectors in the LRN. On the basis of calculation, the value of in-degrees are between 0 and 3 for more than 95% of the nodes. In addition, only 0.004% of the nodes are linked with more than 10 incoming links. Because of applicability of the 80-20 rule or Pareto law in our case, the LRN of the Computer and Internet category is a scale-free network, which is referred to the network with the degree distribution following a power law. In addition, the out-degree distribution of the local relationships is illustrated in Figure 1, pinpointing that only a small number of the weblogs in the Computer and Internet category are linked to the weblogs of the same group. The calculated results showed that 98% of weblogs in the Computer and Internet category link to the 0 to 3 blogs of the same group. Furthermore, only 0.0029% of the weblogs in this category have been linked to more than 10 weblogs of the same group. Hence, the results evidence weak relationships between the Computer and Internet category weblogs.



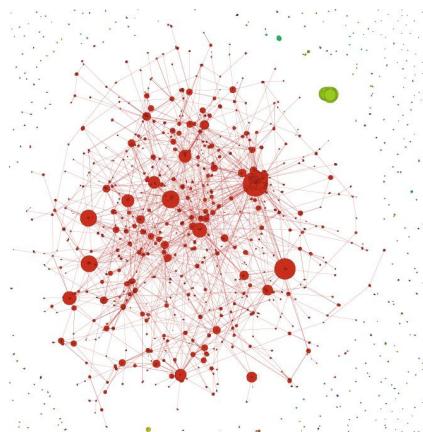
**Fig. 1.** In-degree and Out-degree Distributions of Computer and Internet Category weblogs

## 4.2 Clustering Coefficient

The term Clustering Coefficient (CC) is generally referred to as the probability that two randomly selected friends of the set A (in our case a weblog) are friends with each other [2]. In other words, it can be measured by dividing the number of actual links between one's friends by the number of possible links in full friendship case. Full friendship is the case in which everyone is friend with one another. The value of clustering coefficient is assumed to be between 0 and 1. It is believed that the friends of a person (in our case a weblog) are considered good friends with each other once the CC is close to 1. In other words, if the CC is close to 1, strong relationships between the weblogs can be expected. The calculated average CC for the LRN is as small as 0.006, which is the evidence of no strong relationships between these weblogs.

### 4.3 Connected Components

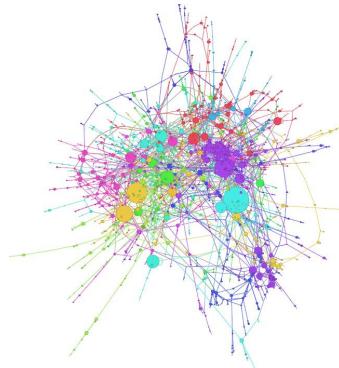
On the basis of the algorithm proposed in [13], the number of weakly and strongly connected components has been calculated in the LRN as 8215 and 15861 components, respectively. Most of the detected components include less than 1% of the nodes and only one giant component existed, that include 43% of the nodes. This giant component is illustrated in Figure 2 using red color. The most remarkable result is the existence of a connected component as shown in green color in Figure 2. This connected component contains 20 weblogs. The principle of homophily indicates that contact between similar individuals occurs more often than among dissimilar individuals [14]. After investigating the content of these weblogs, a content based homophily is detected in the discussed component. That is, the contents provided by these weblogs are related to the mobile technology and general computer learning issues. It can be concluded that the existence of strong relationships among the weblogs of this community originated due to the existence of this content based homophily.



**Fig. 2.** Connected Components in Local relationships

### 4.4 Community Detection by Using Modularity

In order to measure the strength of a network separation into clusters or communities, modularity parameter is proposed in [15]. The number of communities calculated by the modularity is 13083, with the largest one including only 1.55% of the nodes as illustrated in Figure 3 using violet color and the second one including only 1.42% the nodes. Less than 1% of the nodes existed in each of the rest of the communities. It can be inferred from the modularity results that the communications between the Computer and Internet category weblogs are limited to a small number of weblogs which proves our observations reported in previous sections.



**Fig. 3.** Detected Communities in Local Relationship Network network by using Modularity Parameter

#### 4.5 Community Detection by Using K-Core Method

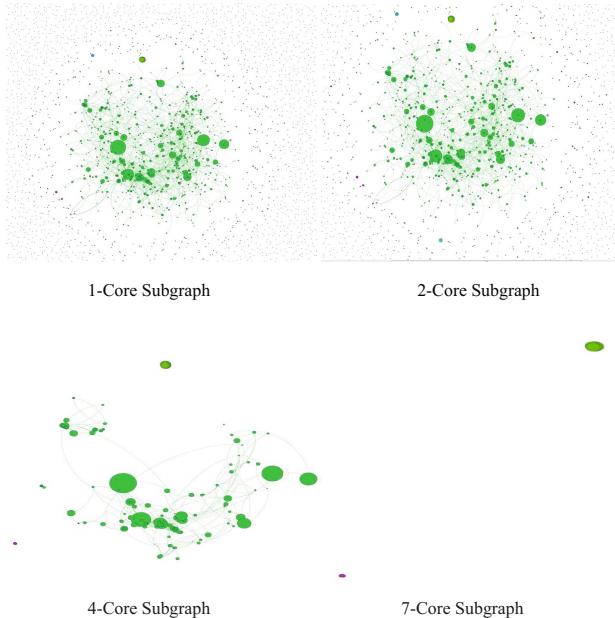
K-core refers to a maximal connected subgraph in which each node is adjacent to at least a minimum number,  $K$ , of the other nodes in the subgraph. In social network analysis, K-core method can be used for community detection. LRN of The Computer and Internet category is drawn based on different values of  $K$  as shown in Figure 4. By increasing the  $K$  value to 4, most of the nodes are discarded from the subgraph and only a small subset of the blogs were found to have relationships with more than 4 nodes. As illustrated in Figure 4, by increasing the value of  $K$  to 7, only 2 communities were remained in K-core subgraph, with one community, equal to the 1 as described in Section 4.3. This community were disappeared by increasing the  $K$  value to 22.

### 5 Number of Incoming Links of the Weblogs and Their Contents

Previous studies have shown that the web pages with the highest in-degree are those providing contents in a vast range of subjects. We investigated this phenomena in blogs of the Computer and Internet category. For instance, the weblog with the address <http://www.biya2it.blogfa.com> has the highest incoming links from the other weblogs. After investigating the contents of the weblog, 37 categories of subjects such as mobile games, web design, movie, health, software, learning computer, etc are detected.

### 6 Analysis of Global Relationship Network (GRN)

In order to analyze the relationships between the Computer and Internet category and the other categories, number of 64000 Blogfa weblogs are crawled and



**Fig. 4.** 1-2-4-7 Core Subgraphs

categorized in 15 categories and 43 subcategories. Then, the number of friends links of the Computer and Internet Category are extracted. Due to the difference in number of weblogs in each category and in order to make fair analysis, the ratio of the Number of Weblogs (NB) to the Number of Friends' links (NF) are investigated. As listed in Table 1, an approximate rate of 88% of the weblogs in the Weblog Themes category together with 18% of the weblogs in the Writing Weblog Issues category are linked by the Computer and Internet category weblogs. Hence, it can be interpreted that there is a common interest between the bloggers to link to the weblogs that provide the contents about installation, maintenance, and configuration of the weblogs. This observation is originated from the fact that the bloggers tends to select a beautiful theme for their weblogs, customize their weblogs and install some applications on them, etc when they create their weblogs for the first time. Consequently, with a high probability, the bloggers visit the weblogs that provide the contents about the mentioned issues and make link to them. In other words, some general subjects are common interests among the most of the bloggers and cause making link to the weblogs that provide contents related to common subjects.

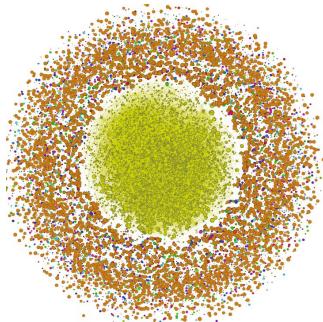
## 7 Visualization and Analysis of the Relationship

The LRN and GRN networks of relationships are illustrated in Figure 5. The number of connected components is equal to 5609 as determined by the Gephi

**Table 1.** The ratio of the Number of Weblogs (NB) to the Number of Friends' links (NF)

| Category                            | Sub-Category                        | NB   | NF  | NB/NF |
|-------------------------------------|-------------------------------------|------|-----|-------|
| News and Media                      | News                                | 796  | 36  | 0.045 |
|                                     | Newspapers and Medias               | 796  | 41  | 0.051 |
|                                     | News writers                        | 796  | 55  | 0.069 |
| Ideology                            | philosophy                          | 1587 | 44  | 0.027 |
|                                     | Islam                               | 1899 | 108 | 0.056 |
|                                     | Quran                               | 1309 | 36  | 0.027 |
|                                     | Christianity                        | 215  | 3   | 0.013 |
|                                     | Zarathustra                         | 225  | 10  | 0.044 |
|                                     | Jewish                              | 96   | 5   | 0.052 |
| Science and Technology              | Health                              | 1770 | 68  | 0.038 |
|                                     | Medicine                            | 1669 | 40  | 0.023 |
|                                     | Nature and Environment              | 1277 | 38  | 0.029 |
|                                     | Foreign Languages                   | 1081 | 75  | 0.069 |
|                                     | Agriculture and Biotechnology       | 1381 | 36  | 0.026 |
|                                     | Electrical and Electronic           | 1880 | 127 | 0.067 |
|                                     | Architecture and Civil Engineering  | 1859 | 55  | 0.029 |
|                                     | Stars                               | 960  | 72  | 0.075 |
|                                     | librarian-ship                      | 392  | 24  | 0.061 |
|                                     | Basic Science                       | 1860 | 88  | 0.047 |
| Culture and History                 | Nurses                              | 325  | 7   | 0.021 |
|                                     | Culture and History                 | 1880 | 90  | 0.047 |
| Persian Speakers in other Countries | Persian Speakers in other countries | 1052 | 42  | 0.039 |
| Fotoblog                            | Fotoblog                            | 1900 | 163 | 0.085 |
| Art and Literature                  | Literature and Poem                 | 1860 | 95  | 0.051 |
|                                     | Book                                | 896  | 88  | 0.098 |
|                                     | theater and Movie                   | 1940 | 97  | 0.05  |
|                                     | Music                               | 1902 | 87  | 0.046 |
|                                     | Imagining                           | 639  | 45  | 0.070 |
|                                     | Artists                             | 1902 | 77  | 0.040 |
| Society and Politics                | Politics                            | 1960 | 39  | 0.019 |
|                                     | Women                               | 400  | 17  | 0.012 |
|                                     | Society                             | 1327 | 30  | 0.022 |
| Family and Life                     | Family and Life                     | 1820 | 81  | 0.044 |
| Tourism and Travel                  | Tourism and Travel                  | 1296 | 48  | 0.037 |
| Personal                            | Personal                            | 1835 | 83  | 0.045 |
| Entertainment                       | Entertainment                       | 1879 | 192 | 0.10  |
| Business and Economic               | Articles                            | 1371 | 43  | 0.031 |
|                                     | Companies and Organizations         | 1880 | 34  | 0.018 |
|                                     | Electronics Commerce                | 1140 | 32  | 0.028 |
|                                     | Earn Money by Internet              | 1231 | 41  | 0.033 |
| Sports                              | Sports                              | 1880 | 103 | 0.054 |
| Weblog Utilites                     | Politics                            | 1960 | 39  | 0.019 |
|                                     | Weblog Themes                       | 377  | 332 | 0.88  |
|                                     | Writing Weblogs Issues              | 237  | 45  | 0.18  |

software. It can be seen that there is a strong connected component at the center of the graph and all nodes in this component have strong relationships with each other. This giant component includes 75% of the nodes. The relationships in the other components are weak. Moreover, the second giant component as shown in Figure 5 using brown color includes only 8% of the nodes.



**Fig. 5.** The LRN and GRN relationships network

## 8 Conclusion

The current paper, presented an analysis of relationships network of weblogs based on their friends' links for the Computer and Internet category of Blogfa Persian weblogs. Some basic analysis based on in-degree, out degree, clustering coefficient, and modularity centrality parameters has been performed and reported. The results from the basis analysis evidenced weak relationships between the weblogs of this group. In addition, the relationship between the number of incoming links of the weblogs and their contents were investigated. The results proved that the weblogs with the highest in-degree are those providing contents in a vast range of subjects. Finally, an analysis on the relationships of the Computer and Internet category of weblogs with other categories has been carried out. This analysis shown that the most of the bloggers tend to link the weblogs that provide the contents about the subjects of common interests among the most of the blogger.

## References

1. Ding, L., Shi, P.: Social network analysis application in bulletin board systems. In: 2011 International Conference on Intelligence Science and Information Engineering (ISIE), pp. 317–320 (August 2011)
2. Easley, D., Kleinberg, J.: Networks, Crowds, and Markets: Reasoning About a Highly Connected World. Cambridge University Press (2010)
3. Abbasi, A., Altmann, J., Hossain, L.: Identifying the effects of co-authorship networks on the performance of scholars: A correlation and regression analysis of performance measures and social network analysis measures. *J. Informetrics* 5(4), 594–607 (2011)

4. Wasserman, S., Faust, K., Iacobucci, D.: Social Network Analysis: Methods and Applications (Structural Analysis in the Social Sciences). Cambridge University Press (November 1994)
5. Ya-ting, L., Jing-min, C.: The social network analysis of political blogs in people: Based on centrality. In: 2011 International Conference on Consumer Electronics, Communications and Networks (CECNet), pp. 5441–5444 (April 2011)
6. Blogfa: Blogfa weblog service provider (2012), <http://www.blogfa.com>
7. Hamulic, I., Bijedic, N.: Social network analysis in virtual learning community at faculty of information technologies (fit), mostar. Procedia - Social and Behavioral Sciences 1(1), 2269–2273 (2009)
8. Qazvinian, V., Rassoulian, A., Shafiei, M., Adibi, J.: A large-scale study on persian weblogs. In: The Proceedings of 12th International Joint Conference on Artificial Intelligence, Workshop of TextLink 2007 (2007)
9. Huiqing, N.: Social network analysis of university online forum. In: 2010 International Conference on Computational Aspects of Social Networks (CASoN), pp. 422–429 (2010)
10. Wen-jun, S., Hang-ming, Q.: A social network analysis on blogospheres. In: 15th Annual Conference Proceedings of the International Conference on Management Science and Engineering, ICMSE 2008, pp. 1769–1773 (September 2008)
11. winwebcrawler: Win web crawler (2012), <http://www.winwebcrawler.com/>
12. Gephi: An open-source graph visualization and maniuplation software (2012), <http://www.gephi.org>
13. Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM J. Comput. 1(2), 146–160 (1972)
14. McPherson, M., Lovin, L., Cook, J.: Birds of a feather: Homophily in social networks. Annual Review of Sociology 27(1), 415–444 (2001)
15. Newman, M.E.J.: Modularity and community structure in networks. Proceedings of the National Academy of Sciences 103(23), 8577–8582 (2006)

# Neighborhood-Based Dynamic Community Detection with Graph Transform for 0-1 Observed Networks<sup>\*</sup>

Li Wang<sup>1,3</sup>, Yuanjun Bi<sup>2</sup>, Weili Wu<sup>2,1</sup>, Biao Lian<sup>1</sup>, and Wen Xu<sup>2</sup>

<sup>1</sup> College of computer science and tech., Taiyuan University of tech., Shanxi ,030024, China

<sup>2</sup> Erik Jonsson School of Engi. and Comp. Science, Univ. of Texas, Dallas, 800 W., U. S.

<sup>3</sup> Institute of Computing Tech., Chinese Academy of Sciences, Beijing, 100190, China

l\_1wang@126.com

**Abstract.** Dynamic complex social network is always mixed with noisy data. It is important to discover and model community structure for understanding real social network and predicting its evolution. In this paper, we propose a novel algorithm NDCCD (Neighborhood-based Dynamic Community Detection with graph transform for 0-1 observed networks) to discover dynamic community structure in unweighted networks. It first calculates nodes' shared neighborhood relationship in a snapshot network and deduces the weighted directed graph; then computes both historic information and current information and deduces updated weighted undirected graphs. A greedy algorithm is designed to find the community structure snapshot at each time step. One evaluation formula is proposed to measure the community similarity. Based on this evaluation, the latent communities can be found. Experiments on both synthetic and real datasets demonstrate that our algorithm not only discovers the real community structure but also eliminates the influence of noisy data for better understanding of real network structure and its evolution.

**Keywords:** dynamic community detection, multi-graph transform, 0-1 observed network, nodes' neighborhood relationship.

## 1 Introduction

We can observe the common principles in nature and human society that individuals are always inclined to those similar one. Based on this principle, some sub groups can be formed. If represented by graph model, those sub graphs are called communities by Newman<sup>[1]</sup>, where nodes are connected tightly within each community and the connection is loose outside the community. Finding community is very important for understanding the characteristics of complex network, discovering latent topology, predicting network evolution and so on. Then, Many novel models are proposed and new applications are exploited. However, most current work focuses on finding static

\* Supported by the Major State Basic Research Development Program of China (973) No. 2013CB329602, the science research foundation for the returned overseas Chinese Scholars, NO. 2010-31, International Collaborative project of Shanxi Province, NO.2011081034, US National Science Foundation (NSF) under Grant no. CNS-1016320 and CCF-0829993.

communities. They ignore the change of observed data over time and the models they discovered always do not obtain the topology changing process and lose some latent information. Some works about dynamic community also proposed recent years. But most of them only calculate snapshot data and compare adjacent snapshot topologies. When there are noisy data in observed dataset, the discovered community structure sometimes maybe wrong.

It is necessary to delve into the dynamic aspects of network behavior, yet it would not be feasible without the data to support such explicitly dynamic analysis. With the development of computer networks and wide applications of social network software, the human society and computer networks fuse much more than before. Many applications on network, such as email, blog, facebook, instant communication, mobile network, sensor networks and so on, supply facile abundant source of dynamic datasets for studying and finding more detailed dynamic topology information in computer network-based social networks.

## 2 Related Work

Recently, some work on identifying dynamic community is published. Most methods focus on evolution clustering and optimization models. Chakrabarti et al.[2] brought forward evolutionary clustering method and its optimization model. They considered that there is relationship between time  $t$  and  $t+1$  and its changing should be smooth. Chi et al.[3] put forward evolutionary spectral clustering algorithm. He extended similarity computing methods and utilized graph cut to measure community structure and community evolution. YU-RU Lin[4] et al. made use of nonnegative matrix factorization, based on Markov probability model and dirichlet distribution to build dynamic community framework Facenet. Chayant et al. [5] enumerated the different situations of community dynamic change and set up different cost evaluation methods that include individual cost, group cost, color cost and built optimization model for dynamic community detecting. Lei Tang et al. [6] proposed a spectral algorithm to model dynamic community evolution.

There are some graph evolutionary algorithms related with dynamic community detection. Kumar et al.[7] researched blog community evolution and burst law based on the change of in-degree, out-degree, strongly connected components and so on. Leskovec et al.[8] analyzed graph evolution models in various fields and proposed generators that produce graphs exhibiting the discovered patterns.

Except evolutionary clustering, graph evolution and optimization models, some methods are put forward recently. Q-k Zhao et al. [9] built vector-based heterogenic networks, extracted the characteristics of network snapshots and network sequence to predict community member. Palla et al [10] analyzed a co-authorship network and a mobile network using the clique percolation method (CPM). Spiliopoulou et al.[11] proposed a framework MONIC to monitor cluster transitions over time. Asur et al.[12] defined a set of events on both individual and community to model community evolution. They defined a set of metrics to calculate the individual and community stability, sociability, influence, popularity and so on. Sun et al.[13] put forward a parameter-free algorithm GraphScope that extracts community and detect community change by Minimum Description Length principle. M. G-R [14] developed an on-line

algorithm for dynamic network inference that relies on stochastic convex optimization. Kumar et al.[15] divided social network into three classes and studied their evolution characteristics. Li [17] built self-organization communities for network management.

There is a common disadvantage for most of above mentioned studies except CPM, that is, they have analyzed direct interaction among individuals and failed to consider the relationship of individual's shared neighbors. In contrast, the CPM considers nodes' neighbor relationship and defines that two k-cliques are adjacent if they share k-1 nodes. However this definition is too rigid such that many small communities cannot be uncovered. In addition, some works took historic information and time dependence into account, but they did not consider the time difference of historic information. Actually the different time that former interaction happened has different influence on current relationships.

In this paper, we put forward a novel algorithm NDCD (Neighbors-based Dynamic Community Detection with graph transform for 0-1 observed network) to identify dynamic communities. The main characteristics are threefold:

1. We discover dynamic communities based on the nodes' relationship of common shared neighbors.
2. When detecting communities at each time, we value the historic and current information. When colligating the two elements, we consider the interval of historic information and calculate the fade away rate of historic information.
3. We define an optimization model for qualifying community at each time step and build a new algorithm for measuring the similarity of community topology. All of these help to find stable communities in suitable granularity.

### 3 Preliminaries and Basic Definitions

#### 3.1 Preliminaries

According to social experiences, some assumptions and preliminaries are built as the following.

1. When we observe the dynamic evolution process, we can see that some context information will show more detailed information of nodes' relationship than just the observed immediate relationship. The relationship of common shared neighbors stays more stable and noisy data has less influence on it than on immediate relationship for identifying topology based on group.
2. In weighted networks, a good community topology is that the weight sum of edges inside all communities is larger than that of edges inter communities.
3. For an active node with many neighbors, each single neighbor's influence is little when deciding the node's community relationship and vice versa.
4. If one relationship between two nodes exists during nonstop time sequence, it implies this close relationship is stable. If one relationship is not observed in continuous times, it means that it is less stable than that in continuous steps.
5. Both historical information and observed snapshot information are important for discovering communities. But their importance is different and the historical information effect will decrease with time.

According to the above five preliminaries, we proposed NDCD algorithm for 0-1 observed graphs. There are three graphs in this algorithm: observed 0-1graph, weighted directed dynamic graph and synthesized undirected weighted graph.

### 3.2 Basic Expressions and Definitions

Firstly we explain the three different graphs in NDCD.

- **Observed graph**  $OG_t = \{V_t, E_t\}$ .  $V_t$  is the node set that can be observed at time  $t$  and  $E_t$  is the edge set that records the interaction or relationship status happening at time  $t$  and  $E_t \in \{0,1\}$ .
- **Weighted directed dynamic graph**  $DG_t$  is a weighted and directed graph that comes from  $OG_t$ . It records the integrated information of social network at timestamp  $t$ .  $DG_t = \{V_t, E_t, R_t\}$ , in which  $V_t$ ,  $E_t$ ,  $R_t$  are node set, edge set and edge weight set that collect all observed graphs from the first time to time  $t$ ;  $R_t$  is a vector  $\{node, neighbor node, synthetic neighbor effectiveness degree\}$ .
- **Synthesized undirected weighted graph**  $DG'_t$  is deduced from  $DG_t$  and the transfer rule will be presented in def. 5.

Then we introduce some important basic definitions.

**Def.1:**  $path(i,j)$  is the min path length between nodes  $i$  and  $j$ . Because the observed snapshot graph is unweighted, the  $path(i,j)$  is the minimum number of nodes in the path that connects  $i$  and  $j$ . If there is an edge  $e(i,j)$  then  $path(i,j)=1$ .

**Def.2:**  $i_{\alpha\text{-neighbor}} = \{k | path(k,i) \leq \alpha, k \in V\}$ . It shows the node's neighbor area, in which  $\alpha$  is a positive integer. The neighbor relationship is symmetric. In this paper we set  $\alpha$  as 1.

**Def.3:** If the observed 1-neighbor area of node  $j$  is  $\{i_1, i_2, i_3, \dots, i_m\}$ , the observed neighbor effectiveness at time  $t$  of node  $i_k$  ( $1 \leq k \leq m$ ) to  $j$  is  $1/m$ , we write it as  $b'_{i_k,j} = 1/m$ ,  $m = |j_{\alpha\text{-neighbor}}|$ .

**Def.4:** If node  $j$ 's  $\alpha$ -neighbor area is  $\{i_1, i_2, i_3, \dots, i_m\}$ , the integrated neighbor effectiveness at time  $t$  of  $i_k$  ( $1 \leq k \leq m$ ) to  $j$  is

$$s'_{i_k,j} = s^{t-1}_{i_k,j} \times (1 - \gamma) + b'_{i_k,j}, s^1_{i_k,j} = b^1_{i_k,j}, i_k \in \bigcup_{q=1} j_{\alpha\text{-neighbor}}^{t_q} \quad (1)$$

$\gamma$  is the information lapsing factor that shows the invalidated percentage of history information with time.

**Def.5: Transfer rules from  $DG_t$  to  $DG'_t$  -----Tran<sup>0-1</sup>:**

$$DG_t(V^t, E^t, R^t) \Rightarrow DG'_t(V'^t, E'^t, R'^t):$$

- (1)  $V^t = V'^t$ ,
- (2) if  $\exists s'_{i,j} \vee s'_{j,i}$ , then  $\exists e'_{i,j} \wedge e'_{j,i}$  and  $e'_{i,j} = e'_{j,i}$
- (3)  $R'^t = R^t + R^{t\perp}$ , that is,  $e'_{i,j} = e'_{j,i} = s'_{i,j} + s'_{j,i}$ ,  
 $e'_{i,j} \in E'^t, s'_{i,j} \in E^t$

**Def. 6:** CC (Community Clustering) shows the tightness of intra nodes in community.

$$CC = \frac{2|E|}{|V|(|V|-1)} \quad (2)$$

$|E|$  is the edge number and  $|V|$  is the node number of this community.

## 4 Neighborhood-Based Dynamic Community Detection

Because our community definition is based on shared neighborhood relationship and we focus on community dynamic changing process, we build the community evaluation methods for social network snapshot and social network sequence.

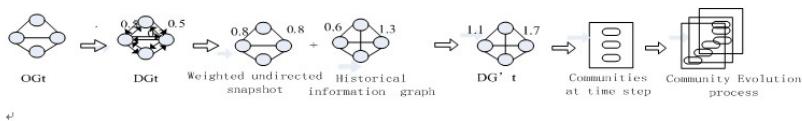
For community snapshot, we set two optimization goals:

- If the weight difference between the sum of intra communities and that of inter communities is the maximum, the community topology is best.
- If one community clustering coefficient CC is larger than given threshold value, we consider this community is good and the community partition should be finished.

For community sequence, we calculate the community difference between adjacent timestamps to decide whether the community structure is stable or not.

NDCD includes six steps as the following.

1. We utilize definition 3 to calculate OGt and get a new matrix  $\text{Neigh}_t$  that records nodes' observed neighbors effectiveness  $b_{i,j}^t$  at time t.
2. Based on  $\text{Neigh}_t$  and definition 4, calculate the nodes' integrated neighbor effectiveness at time t and get the weighted directed dynamic graph DG<sub>t</sub>. Then utilize definition 5 to get the synthetic undirected weighted updated graph DG't.
3. Partition DG't using greedy method and obtain the t-th time community structure.
4. Transfer the obtained community snapshot from weighted graph to unweighted graph. The transfer rule is: there is one beforehand value  $r$  and if edge weight is larger than  $r$ , the edge weight is recorded as 1, otherwise as 0.
5. Calculate each community's CC. If one community's CC is less than given threshold  $\alpha$ , then go to step 3 and partition it further. Repeat this procedure until all communities' CC is larger than  $\alpha$  and get the community structure at time t.
6. Measure the structure similarity of t-th community topology and (t-1)-th community topology. If the similarity is larger than threshold  $\beta$ , then the community topology is stable during this period.



**Fig. 1.** The graph transfer procedure in NDCD

Fig. 1 shows the procedure in NDCD. The detailed is the following:

### 1. Update rules for dynamic graphs $DG_t$

If we observe new nodes, add them to  $DG_t$ . If one node disappears, do not delete it from  $DG_t$  immediately. If we observe new interactions ,update it by def. 4 and 5.

### 2. Algorithm of partition community at one time t

In NDCCD, the observed social network is 0-1 graph and it is transferred to a weighted graph. So finding community snapshot at one time is based on weighted graphs. Santo Fortunato[16] put forward a weighted modularity formula for evaluating the community quality in weighed graphs.

$$Q = \sum_{c=1}^{n_c} \left[ \frac{W_c}{W} - \left( \frac{S_c}{2W} \right)^2 \right], \quad \text{in which } S_c = \sum_{i=1, i \in C}^{|C|} \sum_{j=1, j \in C}^{|C|} s_{ij} \quad (3)$$

$W_c$  is the sum of edge weights in module C; W is the edge weight sum in the network;  $s_{ij}$  is the edge weight that connects i and j.  $S_c$  is the strength sum of vertices in C.  $n_c$  is the number of communities. One optimization goal for community snapshot is to maximize the weighed modularity. According to this goal, we design greedy method to uncover community structure. The basic idea is to remove edges gradually on descent till CC satisfies given threshold.

### 3. Algorithm for measuring the similarity of community structures

In NDCCD, the similarity degree of two communities'  $c_1$  and  $c_2$  is calculated by Jaccard index method as the following.

$$Jac(c_1, c_2) = \frac{|c_1 \cap c_2|}{|c_1 \cup c_2|} \quad (4)$$

Just as introduced before, the matching couples of  $C_t$  and  $C_{t-1}$  that satisfies  $\max \sum_{\substack{1 \leq i \leq |C^{t-1}| \\ 1 \leq j \leq |C^t|}} Jac(c_i^{t-1}, c_j^t)$  are discovered, then the similarity of community topology at different times is computed using the following formula.

$$\Delta_c^t = \sum_{k,h} Jac(c_{i_k}^{t-1}, c_{j_h}^t), \quad \bigcup_k c_{i_k}^{t-1} = C^{t-1}, \quad \bigcup_h c_{j_h}^t = C^t \quad t=1,2,3\dots \quad (5)$$

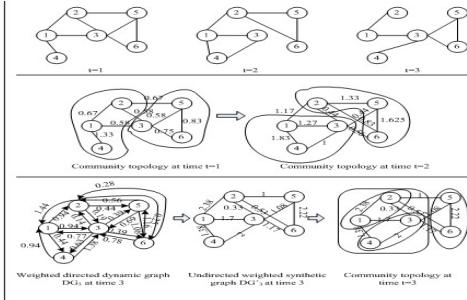
## 5 Experiments and Evaluations

Although some related work is proposed, dynamic community detection is still in its infancy [16] and standard evaluation method is lacked. One popular evaluation way is to compare the identified community structure with real structure; especially it is very useful to analyze the match between identified structure and its semantic content. So

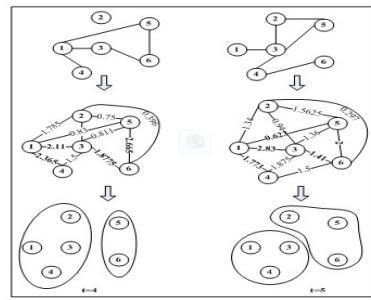
we make experiments on synthetic datasets and real datasets to evaluate NDCD. With synthetic datasets we compare the discovered result with human's knowledge. Based on real datasets we compare the result with its semantic content in real world.

### 5.1 Experiments on Synthetic Datasets

We design a synthetic dynamic network which consists of 6 nodes and give its dynamic changing states in three different time steps. The first line in fig. 2 is the observed networks at different times. The second line is the calculated DG's and community topology at  $t=1$  and  $t=2$ . The third line is the  $DG_t$  and  $DG'_t$  and its hierarchical community snapshot. In this procedure the information lapsing factor  $\gamma = 1/4$ .



**Fig. 2.** The process of NDCD



**Fig. 3.** Demo of NDCD dealing with noisy data

By NDCD, the community structure at time  $t=1$  is  $\{1,2,4\}\{3,5,6\}$ . But by observation, the communities should be  $\{1,2,3\}\{5,6\}\{4\}$ . And the modularity of the later is higher than that of the former. The main difference between them is about the node 3 and 4. For node 4 it will be in the same community with its unique neighbor node 1 in great probability. Node 3 has many neighbors and then these neighbors all have little influence for its topology. But node 3 may affect its neighbors' community attribution greatly. So  $\{5,6\}$  and it belongs to one community. Moreover from the undirected weighted synthetic graph  $DG'_1$ , we can see the weight sum of  $\{3,5,6\}$  is bigger than that of  $\{1,3,2,4\}$ . All of this means that our algorithm accords with the practical experience.

By community similarity function (5), the community similarity between time 1 and 2 is  $1/2$  and that between time 2 and 3 is  $2/3$ . We can see that the community topology is going to be stable during these time steps.

For observing the quality of our algorithm dealing with noisy data, we added two more observed datasets that are respectively at time 4 and 5(Fig.3). At time 4, node 2 is not observed and so it is a noisy data. But by our algorithm we still can compute it and get its community structure as the left in Fig2. The reason is that our algorithm calculates both the historic information and current information. Even there is some wrong data, noisy data, unobserved data in observed snapshot network, its influence could be limited in some scope and we still can get more truthful result.

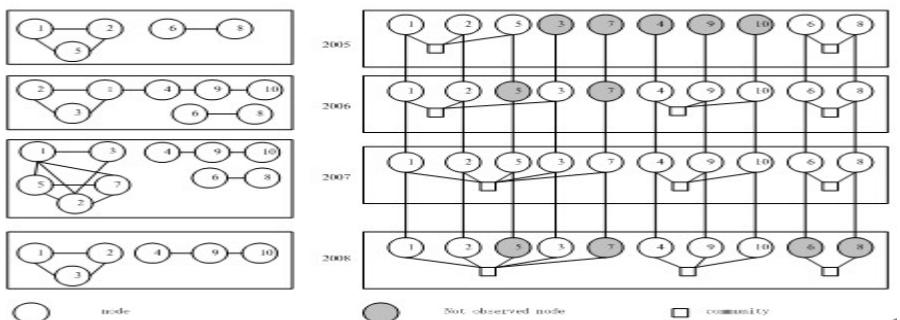
## 5.2 Experiments on Real Dataset

We do experiments on real coauthor networks with two different size (Table 1). We extracted datasets from DBLP and C-DBLP (<http://www.cdblp.cn>). The experiment on little size aims to show NDCD is noise-tolerant and it can discover communities. The one on large size is to show NDCD can discover latent community evolution process.

### 1. Noise-Tolerant Experiment for NDCD

In this test, we extract dataset from C-DBLP. For clearly observing and valuing the discovered community quality, we choose 10 authors and gathered their published papers information from 2005 to 2008. By cleaning and filtering unusable data, we got 31 papers as test dataset. Moreover because we only care about whether there is coauthor relationship between authors, we did not record the number of coauthor papers and hence our observed graph is 0-1 graph. In Fig. 4, the left part shows the observed coauthor relationship and the right part shows the community topology in different years by NDCD. Nodes with circle are authors. In the left part, edge shows that two author nodes collaborated and published papers. In the right part, the dark circled nodes are those authors who are not observed at one time. Square means community. If authors belong to one community, then they connect the same square.

By computing we get the community topology is  $\{\{1,2,3,5,7\}\{6,8\}\{4,9,10\}\}$ . We analyzed the original datasets and found that the affiliations of 1, 2, 3, 5, 7 are same and it is A (for privacy protection), the affiliation of 6 and 8 is B, the affiliation of 4 is C, the affiliation of 9 and 10 is D. The former two communities accord with our experience. Because in China authors in same organizations have more chance to collaborate and publish papers. But why 4, 9, 10 belongs to one community and the affiliation of 4 is different with that of 9 and 10? We researched the original data and found in dataset of 2006 there is one paper whose author is 4 and 1, but the affiliation of node 4 is C. According to our collected other papers by node 4, her affiliation is D. The two authors may be different but with same name or same author but different organizations for short time study or job change. But whether or not, under such situation with part wrong or confused information, NDCD finally discovered the real stable coauthor groups. This shows that dynamic community detecting would eliminate the influence of noisy data well and help to uncover the latent network structure.



**Fig. 4.** Dynamic community of C-DBLP

Analyzing the dynamic community evolution over time, by formula 5 we conclude that the adjacent community topology similarity degrees are 1.68, 2.68 and 3. We can see that the adjacent community topology similarity degrees increase gradually with time and during this observed periods the coauthor community topology tends to be stable.

## 2. Experiment on Community Evolution Discovery

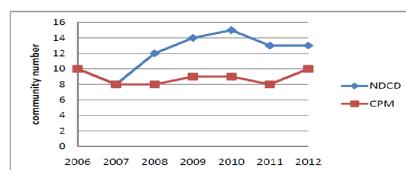
We selected 500 academicians from the four traditional areas: database, network communication, data mining, semantic web and knowledge engineering. From DBLP, We extracted their co-author papers from 2006 to 2012. And we get 43502 coauthor papers. Using NDCD and CPM [10] algorithms respectively, we got communities in each year. For little size communities meaningless, we just analysis the communities with size larger than 20. Fig5 is the comparison between NDCD and CPM.

From NDCD result We can see there are great changes in 2009, 2010 and 2011. We looked into the communities and abstract their papers topic. In 2009, 2010, some old communities were divided and some new communities with little size appeared. From their papers' topic, we found some social network, machine learning related research papers emerged. I think some researchers may turn their interest in social computing and formed new collaboration relationship. In 2011, some communities overlapped and merged. We found more sophisticated topics mixed, such as social influence, machine learning, social media, network analysis. A lot of researchers from different background paid more attention to collaborate and one researcher may have many different research interests. So coauthor communities became bigger.

From CPM, we see the community evolution is slow and nearly stable. That is because they integrated all historical information and snapshot information to update the edge weight. Although they took the decay factor of historical information, but it still influence the result. It makes the evolution stable even if the evolution is dynamic. Our algorithm also takes the decay factor of historical information, but we just integrated the previous one with decay factor. That is consistence with the short period smoothness hypothesis. So our algorithm can help get more reasonable result than CPM .

**Table 1.** Co-author datasets

| dataset     | nodes | edges | duration  | source |
|-------------|-------|-------|-----------|--------|
| Little size | 10    | 31    | 2005~2008 | C-DBLP |
| Large size  | 500   | 43502 | 2006~2012 | DBLP   |



**Fig. 5.** Evolution of community number with year

## 6 Conclusions

In this paper, we proposed a novel algorithm NDCD to identify dynamic communities in observed 0-1 social networks. Differentiated from other algorithms, it leverages the

node relationship of common shared neighbors in observed network snapshots, uncovers community snapshots based on historic information and current observed network and discovers the community evolution process based on graph transfer and evaluation of the adjacent community structures. Experiments on both synthetic and real datasets show that NDCD not only discovers the real community structure but also eliminates the influence of noisy data, thus helps to better understanding of real network structure and its evolution.

## References

1. Newman, M.E.J.: Detecting community structure in networks. *Eur. Phy. J. B* 38 (2004)
2. Chakrabarti, D., Kumar, R., Tomkins, A.: Evolutionary clustering. In: ACM SIGKDD (2006)
3. Chi, Y., Song, X., et al.: Evolutionary spectral clustering by incorporating temporal smoothness. In: Proc. of the 13th ACM SIGKDD Conference (2007)
4. Lin, Y.-R., Chi, Y., Zhu, S., et al.: Facetnet: a framework for analyzing communities and their evolutions in dynamic networks. In: WWW 2008, pp. 685–694 (2008)
5. Chayant, T., Tanya, B.-W., David, K.: A Framework For Community Identification in Dynamic Social Networks. In: KDD 2007, pp. 717–726 (2007)
6. Tang, L., Liu, H., et al.: Community evolution in dynamic multi-mode networks. In: KDD (2008)
7. Kumar, R., Novak, J., Raghavan, P., et al.: On the bursty evolution of blogspace. In: WWW (2003)
8. Leskovec, J., Kleinberg, J., Faloutsos, C.: Graphs over time: densification laws, shrinking diameters and possible explanations. In: KDD (2005)
9. Zhao, Q., Bhowmick, S.S., et al.: Characterizing and predicting community members from evolutionary and heterogeneous networks. In: CIKM 2008, pp. 309–318 (2008)
10. Palla, G., Barabasi, A.-L., Vicsek, T.: Quantifying social group evolution. *Nature* 446 (2007)
11. Spiliopoulou, M., Ntoutsi, I., et al.: Monic: modeling and monitoring cluster transitions. In: SIGKDD (2006)
12. Asur, S., Parthasarathy, S., Ucar, D.: An event-based framework for characterizing the evolutionary behavior of interaction graphs. In: Proc.of the 13th ACM SIGKDD Conf. (2007)
13. Sun, J., et al.: GraphScope: parameter-free mining of large time-evolving graphs. In: KDD (2007)
14. Gomez-Rodriguez, M., Leskovec, J.: Structure and Dynamics of Info. Pathways in Online Media. In: WSDM (2013)
15. Kumar, R., Novak, J., Tomkins, A.: Structure and evolution of online social networks. In: SIGKDD (2006)
16. Fortunato, S.: Community detection in graphs. *Phy. Reports* 486, 75–174 (2010)
17. Wang, L.: SoFA: An expert-driven, self-organization peer-to-peer semantic communities for network resource management. *Expert Systems and Applications* (January 2011)

# Effects of Inoculation Based on Structural Centrality on Rumor Dynamics in Social Networks

Anurag Singh, Rahul Kumar, and Yatindra Nath Singh

Electrical Engineering department, IIT Kanpur-208016, India  
[{anuragsg,rahulkmr,ynsingh}@iitk.ac.in](mailto:{anuragsg,rahulkmr,ynsingh}@iitk.ac.in)

**Abstract.** In social networks, the mechanism to suppress harmful rumors is of great importance. A rumor spreading model has been defined using the susceptible-infected-refractory (SIR) model to characterize rumor propagation in social networks. In this paper a new inoculation strategy based on structural centrality has been applied on rumor spreading model for heterogeneous networks. It is compared with the targeted and random inoculations. The structural centrality of each nodes has been ranked in the topology of social networks which is modeled as scale free network. The nodes with higher structural centrality are chosen for inoculation in the proposed strategy. The structural centrality based inoculation strategy is more efficient in comparison with the random and targeted inoculation strategies. One of the bottlenecks is the high complexity to calculate the structural centrality of the nodes for very large number of nodes in the complex networks. The proposed hypothesis has been verified using simulation results for email network data and the generated scale free networks.

**Keywords:** Complex networks, graph spectra, rumor spreading model, inoculation strategies.

In today's world, Internet has become the most powerful medium to circulate information. We use online social network sites to express our altitude, emotions and to communicate with friends, almost on daily basis. Twitter and Facebook have become the most important mechanisms for information broadcasting. A large number of users share information on them. Consequently, lot of research has been carried out to provide valuable insights in the information diffusion in social networks. If any information is circulated without officially publicized confirmation, it is called a rumor. In other words, rumors are unreliable information.

The rumor spread phenomenon is similar to epidemic spread, in which all the informed nodes spread rumor by informing their neighboring nodes [1,2]. Recent research in complex network theory has given a new direction to the epidemic spreading model [3,4]. It has been found that the topologies of many real world networks have three main properties: small world, scale free and high clustering. The susceptible-infected-refractory (SIR) model for dynamic process of epidemic

spread is used to model the rumor spread in this paper. A susceptible node can be infected by an infected neighbor with some spreading rate and introduces a new refractory state in which nodes cannot be infected. The SIR model for rumor spreading, was first introduced by Daley and Kendal [5] and its variants by Maki-Thomsan [6]. In Daley-Kendal (DK) model, homogeneous population is subdivided into three groups viz., ignorants, spreaders and stifler. The rumor is propagated throughout the population by pairwise contacts between spreaders and other individuals in the population. Any spreader involved in a pairwise meeting attempts to infect other individual with the rumor. In Maki Thomsan (MK) model when a spreader contacts another spreader, only initiating spreader becomes a stifler. DK and MK models have an important shortcoming that these models do not take into account the topology of the underlying social interconnection networks along which the rumors spread. To consider the topology of network, the rumor spreading models on small world network and scale free (SF) networks [7,8] have been defined. Only few studies have been reported on how to stop the rumor spread [9,10,11,12] in small world and SF networks. These studies are more important since false and fatal rumors have negative impacts on the society during disasters.

In this paper, a new inoculation strategy based on structural centrality is applied. Using this method we can find the most influential node in the context of centrality measures in the graph. The structural centrality based inoculation will not be useful for a complex network with very large number of nodes in the complex networks because the complexity will be high in finding the structural centrality of the nodes. In this work, for all the simulations of the complex networks, the scale free property has been considered with power law degree distribution. In real world networks, the scale free properties are found e.g., in email networks, Internet networks, telephone call graphs etc. [4].

## 1 Rumor Spreading Model

In this work, we have used rumor spreading model proposed by us in [8]. We used mean field equations for complex networks while considering non linearly varying number of informed neighbor nodes by a spreader in each time step (not all neighbors of the node).  $P(k) \propto k^{-\gamma}$  is the degree distribution of SF network and  $\Phi(k) = k^\alpha$  is the nonlinear rumor spreading function with  $0 \leq \alpha \leq 1$ .  $P(l|k)$  is the degree-degree correlation function that a randomly chosen edge emanating from a node of degree  $k$  leads to a node of degree  $l$ ,  $P(l|k) = lP(l)/\langle k \rangle$ , for uncorrelated networks, where  $\langle k \rangle$  is the average degree of the network. Let  $I(k, t)$ ,  $S(k, t)$ ,  $R(k, t)$  be the density of ignorants, spreaders and stifler, respectively belonging to connectivity class  $k$  at time  $t$ . The rate equations for rumor diffusion model are,

$$\frac{dI(k, t)}{dt} = -\frac{k\lambda I(k, t)}{\langle k \rangle} \sum_l l^\alpha P(l)S(l, t); \quad (1)$$

$$\frac{dS(k, t)}{dt} = \frac{k\lambda I(k, t)}{\langle k \rangle} \sum_l l^\alpha P(l) S(l, t) - \frac{k\sigma S(k, t)}{\langle k \rangle} \sum_l [S(l, t) + R(l, t)] l^\alpha P(l) - \delta S(k, t); \quad (2)$$

$$\frac{dR(k, t)}{dt} = \frac{k\sigma S(k, t)}{\langle k \rangle} \sum_l [S(l, t) + R(l, t)] l^\alpha P(l) + \delta S(k, t). \quad (3)$$

Where,  $\lambda$ ,  $\sigma$  and  $\delta$  are the rumor spreading, stifling and forgetting rates respectively. After solving equations Eqs. (1)-(3) for  $\delta = 1$ , the rumor threshold (below this spreading rate rumor will not spread in the network) is  $\lambda_c = \frac{\langle k \rangle}{\langle k^{\alpha+1} \rangle}$

## 2 Complex Network Topology Using Graph Spectra

The complex network topology can be understood by the graph structure [13,14]. A graph is defined by  $G = (V, E)$ , where  $V$  is the set of vertices or nodes and  $E$  is the set of edges or links.  $A = [a_{ij}]$  is an adjacency matrix of  $|n \times n|$  size, where  $n = |V|$ ,  $a_{ij}$  will be 1 if edge exists between  $i$  and  $j$  vertices otherwise 0 and  $a_{ij} = a_{ji}$  for undirected (symmetric) graphs. The degree of the  $i^{th}$  vertex,  $d_i = \sum_j a_{ij}$  and  $D = [d_i]$  is the degree matrix which is a diagonal matrix.

Spectral graph theory using eigenvalues and eigenvectors can be applied in the graphs to find out the structural centrality of the graphs. If a matrix is square, symmetric and positive semidefinite [15] then eigenvectors and eigenvalues will exist for the matrix. Eigenvectors and eigenvalues exist for  $A$ , since the adjacency matrix  $A$  of a graph is symmetric, and it is positive semidefinite.

The Laplace matrix,  $L$  of the adjacency matrix  $A$  for graph  $G$  is given by  $L = D - A$ . The Laplace matrix of the graph is a positive semidefinite and symmetric, therefore it has all eigenvalue values, i.e.  $\lambda_i \geq 0, \forall i$ . Hence these eigenvalues ( $\lambda_i$ ) ordered as  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n = 0^2$ , have eigenvectors  $\mathbf{z}_i$  respectively such that  $\|\mathbf{z}_i\|^2 = \mathbf{z}_i^T \mathbf{z}_i = 1$ . The set of eigenvectors of  $L$ ,  $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_i, \dots, \mathbf{z}_n]$ , will be orthonormal i.e.,  $\mathbf{Z}^T \mathbf{Z} = I$ . If  $\Lambda$  is a diagonal matrix,  $\Lambda = [\lambda_{ii}]$  of eigenvalues then  $L$  follows the eigen decomposition as  $L = \mathbf{Z} \Lambda \mathbf{Z}^T$ .

## 3 Structural Centrality

From Laplace matrix  $L$ , Moore-Penrose pseudo inverse matrix  $L^+$  can be defined, that follows all the properties (square, symmetric, doubly-centered, positive semidefinite) of  $L$ . The eigen decomposition of  $L^+$  will be  $\mathbf{Z}^T \Lambda^{-1} \mathbf{Z}$ .  $\mathbf{Z}$  is an orthonormal matrix made of the eigenvectors of  $L^+$ , If  $\Lambda$  has an eigenvalue value,  $\lambda_i = 0$  then corresponding eigenvalue  $\lambda^{-1}$  in  $\Lambda^{-1}$  will also be 0. As  $L^+$  has the doubly centered (all rows and columns sum will be zero) property therefore centroid of the nodes (having position vectors) lies on the origin of the space [15]. The graph matrix maps into the new euclidean space. We can represent each node by a unit vector  $\mathbf{v}$  as,

$$\begin{aligned}\mathbf{v}_i &= [0 \cdots \underset{i}{1} \cdots 0]^T \\ \mathbf{v}_j &= [0 \cdots \underset{j}{1} \cdots 0]^T\end{aligned}$$

Now we can calculate the distance between node  $i$  and  $j$  in terms of number of hops required to reach  $j$  from  $i$  ( $m(j|i)$ ) and vice versa. Average commute hop distance measure is,

$$n(i,j) = m(j|i) + m(i|j) \quad (4)$$

$n(i,j)$  will follow the distance measure for any nodes  $i, j$  and  $k$ ,

1.  $n(i,j) \geq 0$
2.  $n(i,j) = 0$  iff  $i = j$
3.  $n(i,j) = n(j,i)$
4.  $n(i,j) \leq n(i,k) + n(k,j)$

Therefore, using  $L^+$  matrix and graph volume,  $V_G$  ( $= \sum_{k=1}^n d_{kk}$ ),  $n(i,j)$  can be expressed as [15],

$$n(i,j) = V_G(l_{ii}^+ + l_{jj}^+ - 2l_{ij}^+) \quad (5)$$

Now the node vector  $\mathbf{v}_i$  can be mapped into the new euclidean space by using the following transformations,

$$\mathbf{v}_i = Z\mathbf{y}_i, \quad (6)$$

$$\mathbf{y}_i' = \Lambda_i^{1/2} \mathbf{y}_i \quad (7)$$

Where,  $\mathbf{y}_i$  is the transformation node vector. Now Eq. (5) can be decomposed as,

$$\bar{n}(i,j) = V_G(\mathbf{y}_i' - \mathbf{y}_j')^T(\mathbf{y}_i' - \mathbf{y}_j') \quad (8)$$

Hence, in the new euclidean space the node vector  $\mathbf{y}_i$  and  $\mathbf{y}_j$  are separated by average commute euclidean distance measure ( $\bar{n}(i,j)$ ).

Therefore euclidean distance measure for the node  $i$  from the origin can be found as the diagonal entry of the  $L^+$ ,

$$\|\mathbf{y}_i'\|_2^2 = l_{ii}^+ \quad (9)$$

**Definition 1.** If  $L_e$  be the Laplacian of the graph on  $n$  vertices consisting of just the edge  $e$  and  $\mathbf{w} \in \mathbb{R}^n$  then,

$$\mathbf{w}^T L \mathbf{w} = \sum_{e \in E} \mathbf{w}^T L_e \mathbf{w} = \sum_{(i,j) \in E} (\mathbf{w}_i - \mathbf{w}_j)^2 \quad (10)$$

**Definition 2.** Structural centrality is able to make the hierarchy from the most influential nodes to least influential nodes.

The structural centrality of node  $i$  for graph  $G$  is

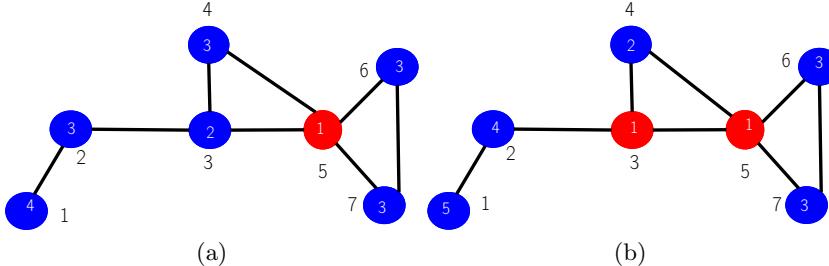
$$SC(i) = \frac{1}{l_{ii}^+}. \quad (11)$$

From Eq. (11), for the lower value of  $l_{ii}^+$  the structural centrality (SC) will be high and vice versa. Therefore the value of  $l_{ii}^+$  is determines the influential nodes.

If a node  $i$  is closer to origin in  $n$ -dimensional space then it will have lower value of  $l_{ii}^+$ , i.e., more centrally located in the network. Therefore, the value of  $l_{ii}^+$  in pseudo inverse matrix  $L^+$  can be defined as,

$$l_{ii}^+ = \sum_{k=1}^{n-1} \frac{z_{ki}^2}{\lambda_k} \quad (12)$$

It has been observed from Eq. (12) that structural centrality of a node is defined by the eigenvectors and eigenvalues of the Laplace matrix,  $L$  of the graph.



**Fig. 1.** The node ranks in graph with (a) degree centralities (b) structural centralities mentioned inside the nodes

The concept of the structural centrality can be understood with the help of an example given in Fig. 1. There are seven nodes in the graph and the hierarchy of their degrees given in the center of the nodes. Hence node 5 is the most influential in the case of targeted inoculation based on nodal degree as shown in Fig. 1(a). After defining the adjacency matrix  $A$  and degree matrix  $D$  of the given graph, we can calculate the Laplace matrix  $L = D - A$ , as

$$L = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 3 & -1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & -1 & 4 & -1 & -1 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & -1 & 2 \end{pmatrix}.$$

Laplace matrix,  $L$  holds following desirable properties to calculate the structural centrality,

1. Symmetric:  $a_{ij} = a_{ji}$ , in  $L$
2. Square matrix :  $L$  is  $7 \times 7$
3. Doubly centered: Summation of all rows and columns in  $L$  is 0
4. Positive semidefinite: Let  $\mathbf{w}$  be any vector, i.e.,  $\mathbf{w} = \begin{bmatrix} -0.8507 \\ -0.5257 \end{bmatrix}$ , then  $\mathbf{w}^T = \begin{bmatrix} -0.8507 & -0.5257 \end{bmatrix}$ , for edge between node 1 and 2,  $L_{12} = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}$ , and

$$\mathbf{w}^T L_{12} \mathbf{w} = 0.3820. \quad (13)$$

Therefore,  $L$  will be positive semidefinite.

Using the  $L$ , pseudo inverse matrix  $L^+$  can be generated as,

$$L^+ = \begin{pmatrix} 1.4626 & 0.6054 & -0.1088 & -0.3469 & -0.4422 & -0.5850 & -0.5850 \\ 0.6054 & 0.7483 & 0.0340 & -0.2041 & -0.2993 & -0.4422 & -0.4422 \\ -0.1088 & 0.0340 & 0.3197 & 0.0816 & -0.0136 & -0.1565 & -0.1565 \\ -0.3469 & -0.2041 & 0.0816 & 0.5102 & 0.0816 & -0.0612 & -0.0612 \\ -0.4422 & -0.2993 & -0.0136 & 0.0816 & 0.3197 & 0.1769 & 0.1769 \\ -0.5850 & -0.4422 & -0.1565 & -0.0612 & 0.1769 & 0.7007 & 0.3673 \\ -0.5850 & -0.4422 & -0.1565 & -0.0612 & 0.1769 & 0.3673 & 0.7007 \end{pmatrix}$$

From the above matrix diagonal values,  $l_{ii}^+$  is defined for  $i^{th}$  node respectively. Thus vector for  $l_{ii}^+ \forall i$  is

$$l_{ii}^+ = [1.462 \ 0.7483 \ 0.3197 \ 0.5102 \ 0.3197 \ 0.7007 \ 0.7007]$$

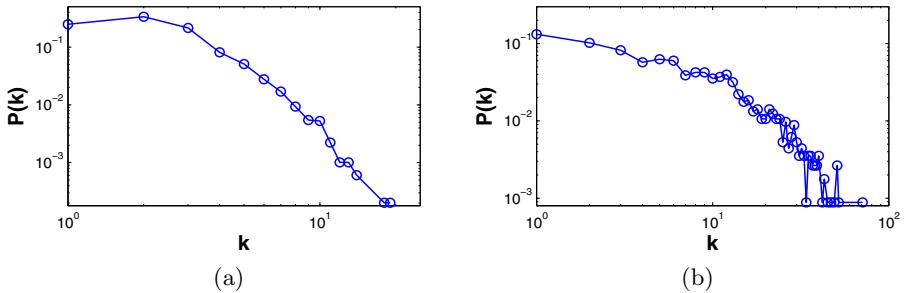
After observing the above values of  $l_{ii}^+$ , it has been found that nodes 3 and 5 have the most structural centrality in the network. Therefore, node 3 can also be most influential like node 5 (i.e. most influential in degree centrality).

## 4 Structural Centrality Inoculations

The diagonal elements,  $l_{ii}^+$  can be sorted from low to high with their node numbers. Now, we will be able to get the list of the nodes sorted according to their degree centralities from high to low from Eq. (11). Then, we can select fraction,  $g$  of inoculated nodes from the sorted array. Therefore, we will be able to inoculate most structurally central nodes first.

## 5 Random Inoculations

In random inoculation strategy, randomly selected node will be inoculated. This approach inoculates a fraction of nodes randomly, without any information about the network. Here, variable  $g$  ( $0 \leq g \leq 1$ ) defines the fraction of inoculated nodes. In the presence of random inoculation, rumor spreading rate  $\lambda$  is reduced by a factor  $(1 - g)$ .



**Fig. 2.** The degree distributions of (a) generated SF network (b) Email network

## 6 Targeted Inoculations

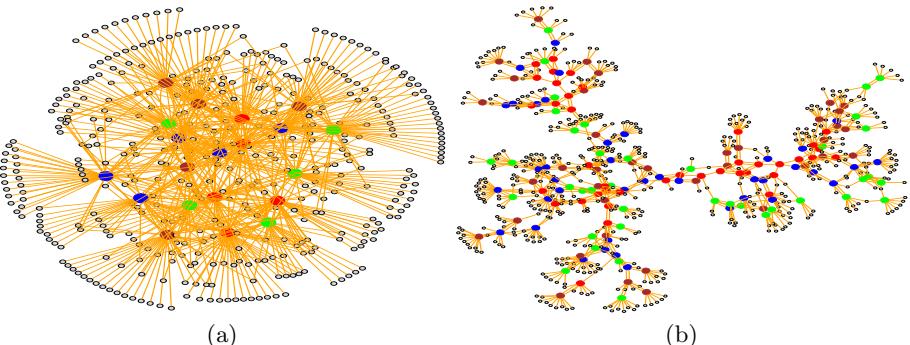
Scale free networks permit efficient strategies which depend upon the hierarchy of the degrees of nodes (degree centrality). The SF networks are strongly affected by targeted inoculation of nodes [8]. In targeted inoculation, the high degree nodes have been inoculated as they are more likely to spread the information. In SF networks, the robustness of the network decreases with a tiny fraction of inoculated individuals.

Let us assume that fraction  $g_k$  of nodes with degree  $k$  are successfully inoculated. An upper threshold of degree is  $k_t$ , so that all nodes with degree  $k > k_t$  get inoculated ( $g_k = 1$ ). Fraction  $g_k$  of nodes with the degree  $k$  are successfully inoculated.

## 7 Simulations and Results

The numerical simulations have been done to observe the complete dynamical process with inoculation strategies with spreading ( $\lambda = 0.5$ ), stifling ( $\sigma = 0.2$ ) and spontaneous forgetting ( $\delta = 1$ ) rates. Nodes interact with each other for rumor passing in each time step. After  $N$  nodes update their states according to the proposed rumor model, time step is incremented. To reduce the complexity,  $\alpha = 1$  is considered. The SF networks are generated according to the power law,  $P(k) = k^{-\gamma}$ , where  $2 < \gamma \leq 3$  for  $N = 5000$  and  $\gamma = 2.3$  (Fig. 2 (a)). Email network has also been considered for the verification as real world complex network (Fig. 2 (b)). The random inoculation is implemented by selecting  $gN$  nodes randomly in the network. The targeted inoculation can be done after selection of the fraction of higher degree of nodes. The structural centrality inoculation can be done by getting the diagonal values,  $l_{ii}^+$  of the pseudo inverse matrix  $L^+$ , for the corresponding node  $i$ . Using  $l_{ii}^+$ , we can sort the values in an array from low to high and inoculate fraction of the sorted nodes in the array.

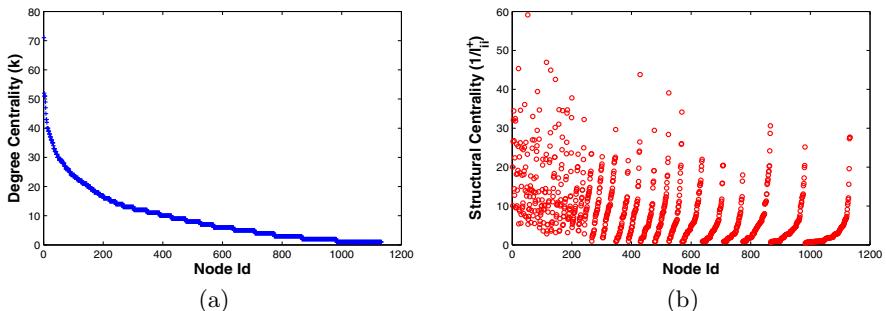
The structures of the email and generated SF network are constructed for some nodes along with the structural centrality (Fig. 3). In the degree distribution of email network more number of very high degree nodes are found as compared



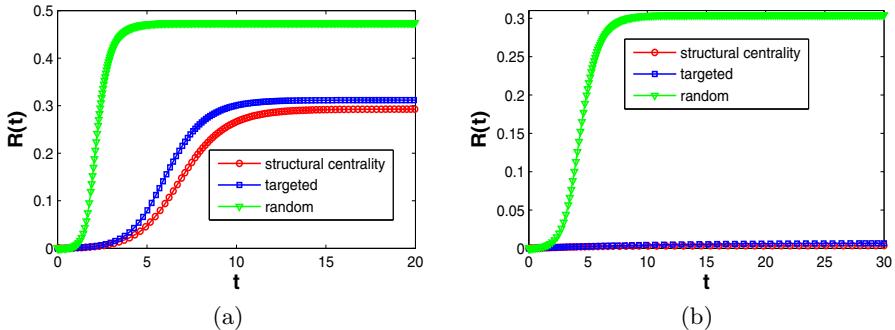
**Fig. 3.** The structure of (a) Email network (b) generated SF network with the different ranking of structural centralities (red → blue → brown → green nodes show higher to low ranking)

with the generated SF network, as shown in Fig. 2. Therefore, Fig. 3 (a) shows lot of edges around more number of higher degree nodes as compared to the generated SF networks shown in Fig. 3 (b). The most structurally central node represented by red and least by green can be verified from Fig. 3. For high structurally central node, less number of hops are required to reach the other nodes, even at less degree. The most structurally centered node provides the well connected path between the two dense nodes shown as a sub-graph.

In Fig. 4 (a) degree centrality has been mentioned for all the nodes in the decreasing order of degrees and corresponding node's structural centrality has been shown as in Fig. 4 (b) for email networks. It is observed that even with the less degree of nodes, the structural centrality is high, and can affect the network in the case of rumor spreading in comparison with the high degree nodes. Therefore, we observe influential nodes in the structural centrality. Hence, it is required to inoculate these nodes to suppress the rumor in the network.

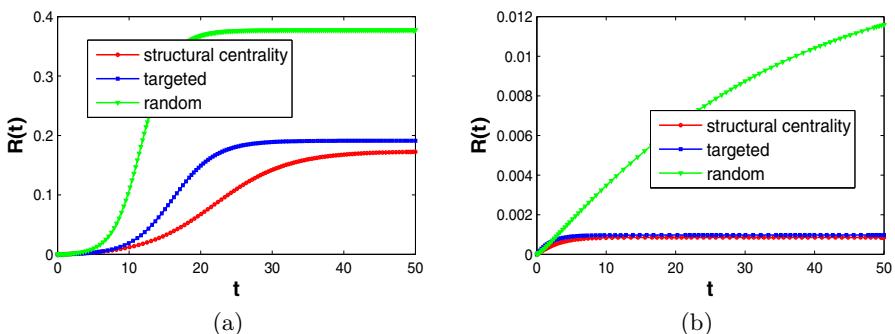


**Fig. 4.** Distributions of (a) degree centralities (b) structural centralities with the node ids in email network



**Fig. 5.** Rumor evolution with the time for (a) 10 % inoculations (b) 30 % inoculations for Email network

Using the rumor model from Eqs. (1)-(3), rumor dynamics is studied for random inoculation, and targeted inoculation on the basis of nodal degree and structural centrality. In Fig. 5, evolution of size of rumor is plotted against time for email network. Final size of rumor is less in the structural centrality than the targeted inoculation for 10 % inoculation of nodes (Fig. 5 (a)). Similar pattern for rumor evolution with time has been found for 30 % inoculations (Fig. 5), but rumor is almost suppressed in this case. Thus, the structural centrality based inoculation suppresses the rumor in the networks more effectively. Random inoculation is not much effective in both cases i.e., in email network and generated SF network to suppress the rumor . In the case of generated SF networks, for very small fraction of time, rumor size has been found to be higher in structural centrality based inoculations initially for 10 % as well as 30 % of inoculations of node, as shown in Fig. 6. But later rumor size decreases in the structural centrality based inoculation in comparison with targeted inoculations (the reason is, highest degree is very less in the network but number of high degree nodes are more). Therefore, degree centrality plays important role initially but later structural centrality plays its role.



**Fig. 6.** Rumor evolution during with the time for (a) 10 % inoculations (b) 30 % inoculations for generated SF network

## 8 Conclusions

In SF network we have derived the structural centralities of nodes in the complex networks and ranked it with the help of  $l_{ii}^+$  values. A node with the high structural centrality needs less number of hops to reach the other node, even at less degree. We have inoculated nodes according to the rank of structural centrality. After this we observed less rumor spreading than the degree centrality based targeted and random inoculations. It is also observed that there are lot of nodes, having low degrees but high structural centralities and vice versa.

## References

1. Moreno, Y., Pastor-Satorras, R., Vespignani, A.: Epidemic outbreaks in complex heterogeneous networks. *Euro. Phy. J. B* 26(4), 521–529 (2002)
2. Zhou, J., Xiao, G., Cheong, S.A., Fu, X., Wong, L., Ma, S., Cheng, T.H.: Epidemic reemergence in adaptive complex networks. *Phys. Rev. E* 85, 036107 (2012)
3. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. *Science* 286(5439), 509–512 (1999)
4. Newman, M.: The structure and function of complex networks. *Siam Review* 45(2), 167–256 (2003)
5. Daley, D., Gani, J., Gani, J.: *Epidemic Modelling: An Introduction*. Cambridge University Press, Cambridge (2001)
6. Maki, D., Thompson, M.: Mathematical models and applications: with emphasis on the social, life, and management sciences. Prentice-Hall, NJ (1973)
7. Nekovee, M., Moreno, Y., Bianconi, G., Marsili, M.: Theory of rumor spreading in complex social networks. *Phy. A* 374(1), 457–470 (2007)
8. Singh, A., Singh, Y.N.: Nonlinear spread of rumor and inoculation strategies in the nodes with degree dependent tie strength in complex networks. *Acta Physica Polonica B* 44(1), 5–28 (2013)
9. Singh, A., Singh, Y.N.: Rumor spreading and inoculation of nodes in complex networks. In: Proceedings of the 21st International Conference Companion on World Wide Web, WWW 2012 Companion, pp. 675–678. ACM (2012)
10. Singh, A., Kumar, R., Singh, Y.N.: Rumor dynamics with acceptability factor and inoculation of nodes in scale free networks. In: 2012 Eighth International Conference on Signal Image Technology and Internet Based Systems (SITIS), pp. 798–804 (November 2012)
11. Singh, A., Singh, Y.N.: Rumor dynamics with inoculations for correlated scale free networks. In: 2013 National Conference on Communications (NCC), pp. 1–5 (2013)
12. Pastor-Satorras, R., Vespignani, A.: Epidemics and immunization in scale-free networks. In: Bornholdt, S., Schuster, H.G. (eds.) *Handbook of Graphs and Networks: From the Genome to the Internet*, pp. 113–132. Wiley-VCH, Berlin (2002)
13. Spielman, D.: Spectral graph theory and its applications. In: 48th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2007, pp. 29–38 (October 2007)
14. Freeman, L.: Centrality in social networks conceptual clarification. *Social Networks* 1(3), 215–239 (1979)
15. Fouss, F., Pirotte, A., Renders, J.M., Saerens, M.: Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering* 19(3), 355–369 (2007)

# A Dominating Set Based Approach to Identify Effective Leader Group of Social Network<sup>\*</sup>

Donghyun Kim<sup>1</sup>, Deying Li<sup>2</sup>, Omid Asgari<sup>1</sup>, Yingshu Li<sup>3</sup>,  
and Alade O. Tokuta<sup>1</sup>

<sup>1</sup> Dept. of Mathematics and Computer Science, North Carolina Central University,  
Durham, NC 27707, USA

{donghyun.kim,atokuta}@nccu.edu, oaliasga@eagles.nccu.edu

<sup>2</sup> Key Laboratory of Data Engineering and Knowledge Engineering, MOE  
School of Information, Renmin University of China, China, 100872  
deyingli@ruc.edu.cn

<sup>3</sup> Dept. of Computer Science, Georgia State University, Atlanta, GA 30303, USA  
yli@cs.gsu.edu

**Abstract.** Very recently, the study of social networks has received a huge attention since we can learn and understand many hidden properties of our society. This paper investigates the potential of social network analysis to select an effective leadership group of a society. Based on our real life observation, we establish three essential requirements for an effective leadership group, namely Influenceability, Partisanship, and Bipartisanship. Then, we formulate the problem of finding a minimum size leader group satisfying the three requirements as the minimum connected  $k$ -core dominating set problem ( $MCkCDSP$ ), and show its NP-hardness. In addition, we introduce an extension of  $MCkCDSP$ , namely  $MCkCDSP-C$ , which assumes the society has a number of communities and requires at least one representative from each community should be in the leadership. At last, we propose an approximation algorithm for a subclass of  $MCkCDSP$  with  $k = 2$ , and show an  $\alpha$ -approximation algorithm of  $MCkCDSP$  can be used to obtain an  $\alpha$ -approximation algorithm of  $MCkCDSP-C$ .

## 1 Introduction

In our lives, the way we think and behave is continuously affected by the opinions and behaviors of our family and friends. Therefore, by studying the relationship among individuals in our society, we can extract information which is potentially very useful for various social decision making processes. Due to the reason, the study of social networks has been received intensive attentions recently.

---

\* This work was supported in part by US National Science Foundation (NSF) CREST No. HRD-0833184 and by US Army Research Office (ARO) No. W911NF-0810510. This research was also jointly supported by National Natural Science Foundation of China under grants 61070191 and 91124001, the National Research Foundation for the Doctoral Program of Higher Education of China under grant 20100004110001.

Given a graph, a *dominating set* (*DS*) is a subset of the nodes in the graph such that all other nodes are adjacent to at least one node in the subset. Recently, several variations of the minimum DS problem, i.e. the problem of computing a minimum size DS, have been appeared in the field of social network analysis. For instance, Feng et al. [1] and Wang et al. [2] are separated studied the problem of computing minimum cardinality DS of a given social network graph such that each node  $u$  outside the subset has at least  $\lceil \frac{\deg(u)}{2} \rceil$  neighboring nodes in the subset, where  $\deg(u)$  is the degree of  $u$  in the graph (fast information propagation problem [1] and minimum *positive influence dominating set* (*PIDS*) problem [2], respectively). This problem is important since with the subset, we can efficiently spread ideas and information within a group (i.e. advertisement). In [3], Dinh et al. introduced the concept of *total positive influence dominating set* (*TPIDS*), which is a more generalized version of PIDS. Over years, most of DS related researches have focused on the relationship between the DS and the rest of the nodes. Consequently, there is largely a lack of attempt to utilize the relationship among the DS nodes to challenge issues in social networking. To the best of our knowledge, only [4] considers a minimum PIDS computation problem with connectivity requirement.

The human history has shown that it has been always difficult to find a good leadership due to many reasons. As we see on TV everyday, it has been always tough for the political leaders to make a consensus among them, adhere to what they believe in, and make the people to follow the decisions they agreed in. More formally, those qualities can be summarized as follows.

- (a) **Bipartisanship:** for any two members in the group, there always has to be a mediator between them if they are not direct collaborators.
- (b) **Partisanship:** each leader should have a sufficient number of collaborators in the group. By being together with political collaborators, a leader will certainly maintain its belief than by being alone.
- (c) **Influenceability:** the leader group should have a good influence over the rest of the members of the society. This property has been considered as the most important quality of a leader in many leadership quality studies.

In this paper, we study a possibility of using social network structure to select an effective leadership of a given society, which is represented as a directed social network graph, in which there is a directed edge from a node  $u$  to another node  $v$  if  $u$  has an influence over  $v$ . While there are many qualities required for an efficient leader group, we focus on the three given above since they are non-debatably essential qualities. Furthermore, we will also try to minimize the size of the group since a smaller one will be more efficient.

**Contributions.** We model the problem of selecting a minimum size leader group from a given social network graph as the *minimum connected  $k$ -core dominating set problem* (*MC $k$ CDSP*). We would like to emphasize that this is one of the few attempts to study a variation of the minimum DS problem with constraints on the relationship among the dominating nodes. We prove the problem is NP-hard and propose an approximation algorithm for a subclass of MC $k$ CDSP with

$k = 2$ . In addition, we introduce an extension of MC $k$ CDSP, namely MC $k$ CDSP-C, which assumes the society has a number of communities and at least one representative from each community should be in the leader group. Finally, we show an  $\alpha$ -approximation algorithm of MC $k$ CDSP can be used to obtain an  $\alpha$ -approximation algorithm of MC $k$ CDSP-C.

**Organizations.** The rest of this paper is organized as follows. Section 2 introduces several important notations and definitions. We introduce our approximation algorithm for MC $k$ CDSP in Section 3. A variation of MC $k$ CDSP, namely MC $k$ CDSP-C, and our strategy to obtain an approximation of MC $k$ CDSP-C is in Section 4. Finally, we conclude this paper in Section 5.

## 2 Notations and Definitions

In this paper,  $G = (V, E)$  represents a social network graph with node set  $V$  and directional edge set  $E$ . For any subset  $D \subseteq V$ ,  $G[D]$  is a subgraph of  $G$  induced by  $D$ . Now, let us introduce several important definitions: Given a graph  $G = (V, E)$  and a positive integer  $k$ , a subset  $D \subseteq V$  is a  **$k$ -core** if for each node  $u \in D$ ,  $u$  is bidirectionally neighboring with at least  $k$  other nodes in  $D \setminus \{u\}$ . Suppose  $D$  is a  $k$ -core in a graph  $G$ . Then, a subgraph of  $G$  induced by  $D$  is not necessarily bidirectionally connected and can be even disconnected. Given a graph  $G = (V, E)$  and a positive integer  $k$ , a subset  $D \subseteq V$  is a **connected  $k$ -core** if (a)  $D$  is a  $k$ -core and (b)  $G[D]$  is connected via bidirectional edges. Given a graph  $G = (V, E)$  and a positive integer  $k$ , a subset  $D \subseteq V$  is a **connected- $k$ -core dominating set (C $k$ CDS)** of  $G$  if (a)  $D$  is a DS of  $G$ , and (b)  $D$  is a connected- $k$ -core.

**Definition 1 (MC $k$ CDSP).** *Given a graph  $G = (V, E)$  and a positive integer  $k$ , the minimum C $k$ CDS problem (MC $k$ CDSP) is to find a C $k$ CDS of  $G$  with minimum cardinality.*

**Lemma 1.** *Consider a graph  $G = (V, E)$  with bidirectional edges only. Then, a subset  $D \subseteq V$  is a connected-1-core if and only if  $G[D]$  is connected.*

*Proof.* First, if  $G[D]$  is connected, then for each  $u \in D$ ,  $\exists v \in D \setminus \{u\}$  such that there is a link between  $u$  and  $v$ . Therefore,  $D$  is a 1-core. Reversely, by the definition of the connected  $k$ -core presented above, the subgraph of  $G$  induced by a connected-1-core in  $G$  is connected. As a result, this lemma is true.

**Theorem 1.** *M $k$ CDSP is NP-hard.*

*Proof.* We show MC $k$ CDSP is NP-hard by showing its special case, in which  $k = 1$  and all edges in  $G$  are bidirectional, is NP-hard. In this special case, given a graph  $G = (V, E)$ , a subset  $D \subseteq V$  is a feasible solution of MC $k$ CDSP if (a)  $D$  is a DS of  $G$ , and (b)  $D$  is a connected-1-core. By Lemma 1, the second condition is equivalent to require  $D$  to be connected. As a result, in this special case, MC $k$ CDSP is equivalent to the *famous minimum connected dominating set (CDS)* problem in general graphs, which is a well-known NP-hard problem [5]. As a result, MC $k$ CDSP is NP-hard and this theorem is correct.

**Algorithm 1. MC2CDSA ( $G = (V, E)$ ,  $k = 2$ )**


---

```

1: /* Phase 1 starts here. */
2: Build  $G' = (V', E')$  such that  $V' \leftarrow V$  and  $E'$  has only bidirectional edges in  $E$ .
3: Apply Guha and Khuller's 2 stage greedy algorithm [6] to  $G'$  and obtain a CDS  $D$ .
4: Color all nodes in  $D$  in white.
5: For each white node  $u \in D$ , color  $u$  in black if  $\exists v, w \in D \setminus \{u\}$  such that  $u$  is
   connected to  $v$  and  $w$  via bidirectional edges, respectively.
6: while there is a white node  $u \in D$  do
7:   Find a path  $P \subset V' \setminus D$  with length at most 3 hops from  $u$  to another node
    $v \in D \setminus \{u\}$ . Set  $D \leftarrow D \cup P$ , and color  $u, v$  and all nodes in  $P$  in black.
8: end while
9: /* Phase 2 starts here. */
10: Construct a bipartite graph  $B = (V_L, V_R, E_B)$  such that  $V_L \leftarrow V' \setminus D$ ,  $V_R \leftarrow V \setminus V'$ ,
    and for each pair of  $u \in V_L$  and  $v \in V_R$ ,  $E_B \leftarrow (u, v)$ , a directional edge from  $u$  to
     $v$  only if  $(u, v) \in E$ .
11: Apply a set-cover greedy algorithm on  $B$  and obtain a subset  $L \subseteq V_L$ .
12: for each node in  $w \in L$  do
13:   if  $w$  is connected to only one node in  $D$  in  $G'$  then
14:     Find a node in  $u \in L$ , which is neighboring to  $w$  in  $G'$ .
15:     Add  $u$  and  $w$  to  $D$ .
16:   end if
17: end for
18: Output  $D \cup L$ .

```

---

### 3 A New Approximation Algorithm for MC2CDSP

In this section, we propose the *minimum connected 2-core dominating set algorithm (MC2CDSA)*, an approximation algorithm for MC2CDSP, a special case of MC $k$ CDSP with  $k = 2$ . We assume that  $G$  contains a feasible solution of the problem. Given a graph  $G = (V, E)$ , suppose  $G' = (V', E')$  is a subgraph of  $G$  that we can obtain after we remove all directional edges from  $G$ . Note that if  $G$  includes a feasible solution of MC2CDSP, (a) there exists only one  $G'$ , and (b) for each node  $u \in V \setminus V'$ , there exists  $v \in V'$  such that  $(v, u) \in E$ , i.e. there exists a directional edge from  $v$  to  $u$ . Largely, MC2CDSP consists of two phases. In the first phase, MC2CDSP adopts Guha and Khuller's 2 stage  $(3 + \ln \Delta)$ -approximation algorithm [6] for the minimum *connected dominating set (CDS)* problem to  $G'$  and obtain a CDS  $D$  of  $G'$ . Next, we initially color all the nodes in  $D$  in white, but we color those nodes having at least two neighbors in  $D$  in black. Then, until no white node left in  $D$ , we repeatedly (a) pick a white node  $u \in D$ , and find a path  $P$  in  $G'$  with length at most 3 hops from  $u$  to another node  $v \in D \setminus \{u\}$  such that  $P \cap D = \emptyset$ , and (b) add the nodes in  $P$  to  $D$ . Color  $u, v$ , and all nodes in  $P$  in black.

Once the first phase is done,  $D$  becomes a connected-2-core. However, it is possible that there exist some nodes in  $V \setminus V'$  not dominated by  $D$ . To make  $D$  to be a DS of  $G$ , we need to add more nodes from  $V'$  to  $D$  such that  $D$  is still a connected-2-core and all nodes in  $V \setminus V'$  are dominated by some nodes

in  $D$ . For this purpose, we first construct a bipartite graph  $B = (V_L, V_R, E_B)$  such that  $V_L$  is the set of nodes in  $V' \setminus D$ ,  $V_R$  is the set of nodes in  $V \setminus V'$ . At last, establish an edge from each node  $u \in V_L$  to another node  $v \in V_R$  in  $E_B$  only if there exists a directional edge from  $u$  to  $v$  in  $G$ . Then, we apply a well-known set-cover greedy approximation algorithm with performance ratio  $H(\Delta)$  on  $B$ , where  $H$  is a harmonic function. Once a subset  $L$  of  $V_L$  is selected by the algorithm, for each node in  $w \in L$ , we check if  $w$  is bidirectionally connected to another node  $w' \in L$  in  $G$ . Otherwise, we select another node in  $V_L \setminus L$  such that  $D \cup L$  remains to be a connected-2-core. As a result,  $D \cup L$  is a feasible solution of the MC2CDSP instance.

**Theorem 2.** *The output of MC2CDSA is a feasible solution of MC2CDSP.*

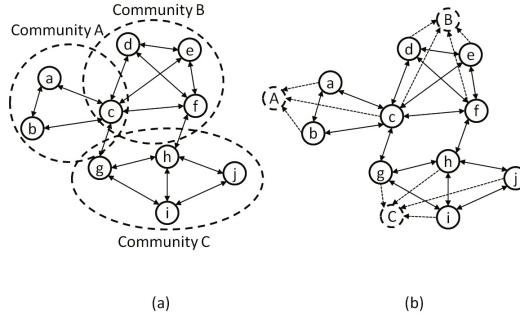
*Proof.* From Line 2-3, we obtain a CDS  $D$  of  $G'$ . Then, each node  $u \in D$  has to have at least one or more neighbors in  $D$  since  $G'[D]$  is connected. Suppose  $u$  has only one neighbor in  $G'[D]$ . Then, there has to be another node  $v \in V' \setminus D$  connected to  $u$  in  $G'$ . Otherwise,  $u$  has no neighbor in  $G'$ , which implies that  $G$  has no feasible solution. Also, notice that  $v$  has to be adjacent at least one neighbor in  $G'[D]$ . Therefore, if we add  $u$  and  $v$  to  $D$ , then each of  $u$  and  $v$  has at least two neighbors in  $D$ . By repeating this process for all nodes like  $u$  in  $D$ ,  $D$  will eventually become a connected 2-core. Clearly, each node  $u \in V' \setminus D$  has to be adjacent to at least one node  $v \in D$ , otherwise,  $D$  is not a dominating set of  $G'$ . However, there exist some nodes in  $V \setminus V'$  which are not dominated by  $D$ . To dominate them, we need to add more nodes from  $V' \setminus D$  to  $D$ . The minimum number of the nodes can be approximated by solving the set-cover problem over the bipartite graph  $B$  induced in Line 10. Unfortunately, once the set  $L$  is constructed by the algorithm,  $D \cup L$  is not necessarily a 2-core even though it is a connected 1-core dominating set of  $G$ . To ensure the 2-core-ness of  $D \cup L$ , we need to add more nodes for those in  $L$ . (Line 13-17). In detail, for each node  $u \in L$ , if it has only neighbor in  $D$ , find another node  $v \in V' \setminus (L \cup D)$  connected to  $u$ . We can prove such a  $v$  does exist all the time as long as there is a feasible solution of the problem in  $G$  using the similar argument given above. As a result,  $L \cup D$  is a connected 2-core dominating set of  $G$ , and this theorem is true.

**Theorem 3.** *The performance ratio of MC2CDSA for MC2CDSP is  $3 \cdot (1 + 2\Delta) \cdot (3 + \ln \Delta)$ .*

*Proof.* Suppose  $OPT$  is an optimal solution of MC2CDSP. In the first phase, we adopt a Guha and Khuller's 2 stage  $(3 + \ln \Delta)$ -approximation algorithm [6] to obtain a CDS  $D$  of  $G'$ . As we proved in Theorem 1, a CDS of  $G'$  is also a connected 1-core, we have  $|D| \leq (3 + \ln \Delta)|OPT|$ . Next, to make  $D$  a connected 2-core, we add a series of paths with length at most three. Therefore, for each node in  $D$ , we add at most 2 more nodes. As a result, we have

$$|D| \leq (3 + \ln \Delta)|OPT| + 2 \cdot (3 + \ln \Delta)|OPT| = 3 \cdot (3 + \ln \Delta)|OPT|.$$

In the second phase, we add more nodes  $D$  so that  $D$  can dominate the nodes in  $V \setminus V'$ . Note that the size of  $L$  is bounded by the size of  $D$  constructed



**Fig. 1.** These figures illustrate an graph conversion example from MCkCDSP-C problem instance (Fig.(a)) to MCkCDSP problem instance (Fig.(b))

so far multiplied by the maximum degree of  $G'$ ,  $\Delta$ , and thus we have,  $|L| \leq \Delta \cdot 3 \cdot (3 + \ln \Delta)|OPT|$ . Furthermore, for each node in  $L$ , we may need to add one more nodes to make  $L \cup D$  a 2-core. Therefore, we have

$$|D \cup L| \leq 3 \cdot (3 + \ln \Delta)|OPT| + (1+1)\Delta \cdot 3 \cdot (3 + \ln \Delta)|OPT| = 3 \cdot (1+2\Delta) \cdot (3 + \ln \Delta)|OPT|,$$

and thus this theorem is true.

## 4 MCkCDSP under Sub Community Structures

Previously, we introduced MCkCDSP whose goal is to find a connected  $k$ -core dominating set of a given social network. In the real application domain, the objective of this problem is to elect a representative group which can effectively operate. However, the formulation ignores one important aspect of our social situation that there are various underlying community structures in our society. Suppose we want to solve MCkCDSP in a way that at least one representative from each community is included in the representative group, which is likely to happen in real world situation. Then, any algorithm for MCkCDSA is not useful anymore. Therefore, in this section, we introduce a variation of MCkCDSP, to deal with this new challenge. We first introduce a variation of MCkCDSP to formulate this problem and show how it can be solved using our result so far.

**Definition 2 (MCkCDSP-C).** *Given a directional social network graph  $G = (V, E)$ , a collection  $\mathcal{C} = \{C_1, C_2, \dots, C_l\}$  of the subsets of  $V$ , and a positive integer  $k$ , the minimum  $Ck$ CDS problem with communities (MCkCDSP-C) is to find a  $Ck$ CDS  $D$  of  $G$  with minimum cardinality such that for each subset  $C_i \in \mathcal{C}$ ,  $C_i \cup D \neq \emptyset$ .*

**Corollary 1.** *MCkCDSP-C is NP-hard.*

*Proof.* The proof of this corollary naturally follows from Theorem 1 since MCkCDSP-C with  $\mathcal{C} = \emptyset$  is equivalent to MCkCDSP.

Given an MCkCDSP-C instance  $\langle G = (V, E), \mathcal{C}, k \rangle$ , we first induce a graph  $G' = (V', E')$  from a given MCkCDSP instance  $\langle G = (V, E), \mathcal{C}, k \rangle$  as follow.

- (a) Copy  $G$  to  $G'$ , i.e.  $V' \leftarrow V$  and  $E' \leftarrow E$ .
- (b) For each subset  $C_i \in \mathcal{C}$ , add a node  $c_i$  to  $V'$ , and add an edge from each node  $u \in C_i$  to  $c_i$  to  $E'$ .

One example of this graph induction is shown in Fig. 1, e.g.  $G$  is in Fig. 1(a) and  $G'$  is in Fig. 1(b). Then, we have the following lemma.

**Theorem 4.** *There exists a feasible solution of an MCkCDSP-C instance  $\langle G, \mathcal{C}, k \rangle$  if and only if there exists a feasible solution of an MCkCDSP instance  $\langle G', k \rangle$ .*

*Proof.* We first show that a feasible solution of an MCkCDSP-C instance  $\langle G, \mathcal{C}, k \rangle$  is a feasible solution of an MCkCDSP instance  $\langle G', k \rangle$ . Suppose  $D$  is a feasible solution of the MCkCDSP-C instance  $\langle G, \mathcal{C}, k \rangle$ . Then,  $D$  is clearly a connected  $k$ -core and dominating all nodes in  $V$  in  $G$ , which implies that  $D$  is a connected  $k$ -core and dominating all nodes in  $V' \setminus \{c_1, \dots, c_l\}$ . By the definition of MCkCDSP-C, for each  $C_i \in \mathcal{C}$ , at least one node in  $C_i$  is included in  $D$ . In addition, by the construction of  $G'$ , each  $c_i$  is dominated by all nodes in  $C_i \subset V'$ . Therefore, for each  $c_i$ , there exists at least one node in  $D$  dominating  $c_i$  in  $G'$ . Next, we show that a feasible solution of an MCkCDSP instance  $\langle G', k \rangle$  is a feasible solution of an MCkCDSP-C instance  $\langle G, \mathcal{C}, k \rangle$ . Suppose  $D'$  is a feasible solution of the MCkCDSP instance  $\langle G', k \rangle$ . Then, for each  $c_i$ , there exists a node  $u \in D'$  dominating  $c_i$  by the definition of MCkCDSP. This means that for each  $C_i \in \mathcal{C}$ , there exists a node from  $C_i$  in  $D'$  in  $G$  by the construction of  $G'$ . Furthermore,  $D'$  is a connected  $k$ -core and dominating all nodes in  $V'$ , which means that  $D'$  is dominating all nodes in  $V$ . As a result, this theorem is true.

**Theorem 5.** *There exists an  $\alpha$ -approximation algorithm for MCkCDSP-C in  $G$  if and only if there is an  $\alpha$ -approximation algorithm for MCkCDSP in  $G'$ .*

*Proof.* Clearly, the cost of a feasible solution  $D$  of an MCkCDSP-C instance  $\langle G, \mathcal{C}, k \rangle$  is equivalent to the cost of  $D$  for an MCkCDSP instance  $\langle G', k \rangle$  since in both problems, the cost of  $D$  is its size, i.e. the cardinality of  $D$ . By Theorem 4, an optimal solution  $O$  of MCkCDSP-C in  $G$  is a feasible solution of MCkCDSP in  $G'$ . Now, suppose  $O$  is not an optimal solution of MCkCDSP in  $G'$ , and there exists another optimal solution  $O'$ . Then, by Theorem 4,  $O'$  is also a feasible solution of MCkCDSP-C in  $G$ . Since this contradicts to our initial assumption that  $O$  is an optimal solution of MCkCDSP-C in  $G$ , such a  $O'$  cannot exist. Therefore, an optimal solution  $O$  of MCkCDSP-C in  $G$  is an optimal solution of MCkCDSP in  $G'$ . Now, suppose we have an  $\alpha$ -approximation algorithm  $\mathcal{A}$  for MCkCDSP-C in  $G$ . Then, an output  $o$  of  $\mathcal{A}$  satisfies  $|o| \leq \alpha|OPT| = \alpha|OPT_1|$ , where  $OPT$  is an optimal solution of MCkCDSP-C in  $G$  and  $OPT_1$  is an optimal solution of MCkCDSP in  $G'$ . As a result,  $\mathcal{A}$  is also an  $\alpha$ -approximation algorithm for MCkCDSP in  $G'$ . Using similar argument, we can prove an output  $o$  of an

$\alpha$ -approximation algorithm for MC $k$ CDSP in  $G'$  is also an  $\alpha$ -approximation algorithm for MC $k$ CDSP-C in  $G$ . As a result, this theorem is true.

## 5 Concluding Remarks and Future Work

In this paper, we study MC $k$ CDSP, a new interesting optimization problem in social networks, and its variation MC $k$ CDSP-C. After we show MC $k$ CDSP is NP-hard, we introduced an approximation algorithm of MC $k$ CDSP with  $k = 2$ . Furthermore, we also prove MC $k$ CDSP-C is NP-hard and show an  $\alpha$ -approximation algorithm of MC $k$ CDSP can be used to have an  $\alpha$ -approximation algorithm of MC $k$ CDSP-C. Meanwhile, in this paper, we were only able to  $\alpha = O(\Delta \ln \Delta)$  with  $k = 2$ , where  $\Delta$  is the maximum node degree of  $G'$ , a subgraph of an input social network  $G$  with only bidirectional edges left. Since it is not difficult to show that a special case of MC $k$ CDSP is equivalent to the set-cover problem (as appeared in Algorithm 1), the performance ratio achievable would be at best  $O(\ln \Delta)$ . Therefore, there is a significant gap between the lower bound of the performance ratio and what we achieved. As a future work, we plan to further study to reduce this gap and will investigate approximation algorithm for general  $k > 2$ .

## References

1. Zou, F., Zhang, Z., Wu, W.: Latency-bounded Minimum Influential Node Selection in Social Networks. In: Proc. of Workshop on Social Networks, Applications, and Systems (2009)
2. Wang, F., Camacho, E., Xu, K.: Positive Influence Dominating Set in Online Social Networks. In: Du, D.-Z., Hu, X., Pardalos, P.M. (eds.) COCOA 2009. LNCS, vol. 5573, pp. 313–321. Springer, Heidelberg (2009)
3. Dinh, T.N., Shen, Y., Nguyen, D.T., Thai, M.T.: On the Approximability of Positive Influence Dominating Set in Social Networks. Journal of Combinatorial Optimization (JOCO) (2012)
4. Zhu, X., Yu, J., Lee, W., Kim, D., Shan, S., Du, D.-Z.: New Dominating Sets in Social Networks. Journal of Global Optimization (JOGO) 48(4), 633–642 (2010)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, San Francisco (1978)
6. Guha, S., Khuller, S.: Approximation Algorithms for Connected Dominating Sets. Algorithmica 20, 374–387 (1996)

# Using Network Sciences to Evaluate the Brazilian Airline Network

Douglas Oliveira<sup>1</sup>, Marco Carvalho<sup>1</sup>, and Ronaldo Menezes<sup>2</sup>

<sup>1</sup> Florida Institute of Technology  
Department of Computer Sciences  
Melbourne, Florida, USA

doliveira2011@my.fit.edu, mcarvalho@cs.fit.edu

<sup>2</sup> Florida Institute of Technology  
Department of Computer Sciences  
BioComplex Laboratory  
Melbourne, Florida, USA  
rmenezes@cs.fit.edu

**Abstract.** In the next few years, Brazil will host international events such as the 2014 FIFA World Cup and the 2016 Olympics Games. Given the worldwide appeal of these events, local authorities in Brazil expect the country will have around 1.6 million visitors arriving at its airports (1 million for the Olympics and 600 thousand for the World Cup). Therefore, these events will put to test the robustness of the Brazilian airline transportation system. As of today, Brazil concentrates most of its flights in and out a single hub city: São Paulo Airport in Guarulhos (GRU). Is this concentration a problem? Aiming to analyze the hub choices of this network we collected data from the five biggest companies that operate in Brazil with domestic and international flights; together these companies are responsible for more than 94% of the total flight traffic in Brazil. In this paper analyzed the impact of moving today's main hub, Guarulhos Airport in São Paulo (GRU), to other airports around the country—the idea is to understand what is the best configuration for single-hub model in Brazil. We also investigated the robustness of the network having a single hub by analyzing the impact of the removal of this hub from the network. We believe this work may help us understand how the airport infrastructure in Brazil has to be developed in the near future.

## 1 Introduction

Air transportation has become vital to tourism and business because, despite its inconveniences, it is still the best and most secure means of travelling long distances. Moreover, air transportation has become cheaper over the years while the security has increased [1]. Unlike some other means of transportation, air transportation is quite organized and it involves many specific institutions, like airline companies, regulatory agencies, and airports that behave accordingly to the market conditions [2]. The analysis of flight routes is important to understand the social, economic, and political causes that led to the formation of the

network. The analysis may also allow for better planning of current and future airline routes helping airlines to be better positioned in the market [3].

Until 1999, the *Empresa Brasileira de Infraestrutura Aeroportuária* (Infraero), a Brazilian Governmental Corporation responsible for operating the main Brazilian commercial airports, was managed by people with military backgrounds from the Brazilian Air Force. In 1999 the Brazilian government named a manager from the Brazilian National Economic and Social Development Bank (BNDES) as president of Infraero [4]. The manager had large experience in privatization in many sectors of the economy. His nomination represented a paradigm shift in the way the corporation was managed and an indication of the intent to privatize public airports [5]. The new administration was structured to prepare Infraero to attract private interest through securing greater efficiency and thus a higher market value.

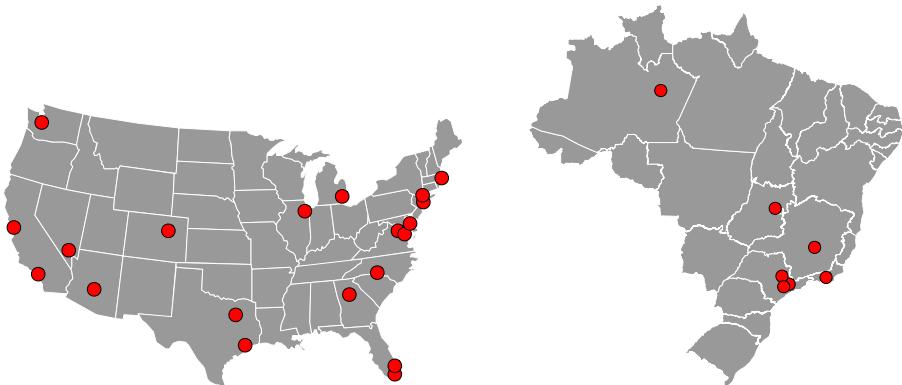
The breakdown of barriers for new airlines to enter into the market and deregulation of the Brazilian aviation industry started in the 90s, leading to an increase of competitiveness. This new market behavior induced the companies to have a business orientation rather than a purely operational one [6]. Hence the actual flight network might not have optimal topology if we consider flights features like the minimum flight time.

If one looks at the airline network in Brazil we can easily notice that it is not organized around hubs as in other large countries such as the United States and China(see Figure 1 for the location of the hubs in the USA and Brazil). This is quite unique given the continental size of the country. In Brazil, most international routes fly to and from to São Paulo Airport (GRU); a single hub system. In this paper we evaluate this one-hub model by moving the hub to other parts of the country and as well as the impact the network after the removal of hubs from the system—one should expect that the more distributed is the system the less susceptible it is to failures.

## 2 Related Work

There have been many works published analyzing flights routes; Burghouwt et al. [7] show the evolution of flight routes in Europe and North America. In contrast to the flight network in the USA, the European network is characterized by a trend for spatial decentralization. A more general picture of the airports network is given by Guimera et al. [8] who models the world-wide airport network and claims that the network is small-world (as expected) and the most connected cities are not the central ones, this happens due to the multi-community structure of the network [9].

In the work of Costa et al. [4], they try to analyze the Brazilian network using an US Federal Aviation Administration measure, which was not well suited because the Brazilian network is quite different from the American one. The Brazilian airport network was also analyzed by Pinheiro and Mello [10] to rank Brazilian airports using optimization methods which take into consideration passengers' movement, the amount of cargo and the number of airplanes. Many



**Fig. 1.** Major airport hubs in the USA (left). Note how well distributed the hubs are around the country. On the other hand we have Brazil (right), a country that shows a concentration of large airports in the south of Brazil. This “centralization” may lead to longer travel times for people who want to travel to the north or northeast of the country.

works analyze the efficiency of the hubs of Brazilian network; one example is the work of Pacheco and Fernandes [11] that focus on airport’s ability to generate financial returns. These works only evaluate a small part of the entire network of flights in Brazil, that is the dataset of flights they use is far from the total number of flights available in the country.

Similar works have analyzed the flight network in a country scale, like Guida and Maria [12] who analyze the topological properties of the Italian airport network. Their findings corroborate with the idea that an airport network is small world and also follows a power law degree distribution. Another work that evaluates an airport network of a country was done by Bagler [13] in the context of the Indian network. His findings are quite similar with the last one, but they also claim that the Indian network is hierarchical and has disassortative properties (i.e. small airports tend to connect to big ones and vice-versa). The Chinese airport network is investigated by Zhang et al. [14] with a focus on the evolution of the traffic flow of both passengers and cargos. Correlated with the annual growth of the country, the traffic continues to grow (in an exponential form) and has evident seasonal fluctuations. A comparative analysis can be found in [15], which tries to identify the most beneficial airport passengers should chose in China, Europe and US to improve their travel time.

Even if we find an optimal airport that could be a hub, papers in the literature as the one of Endres [16] show the consequences of overloading an airport. The work discusses the two worst air-transportation accidents in Brazil (with 354 fatalities) and how their repercussion to Brazilian tourism lasted to the next years. In that period the Brazilian flight system faced problems like lack of confidence not only to safety but also to quality (punctuality). What lead us evaluate the impact of a removal of certain hubs in the Brazilian airline network

(as if an accident happened) is our belief that a network with a few hubs can be very risky for the entire air transportation system.

### 3 Data Collection and Network Analysis

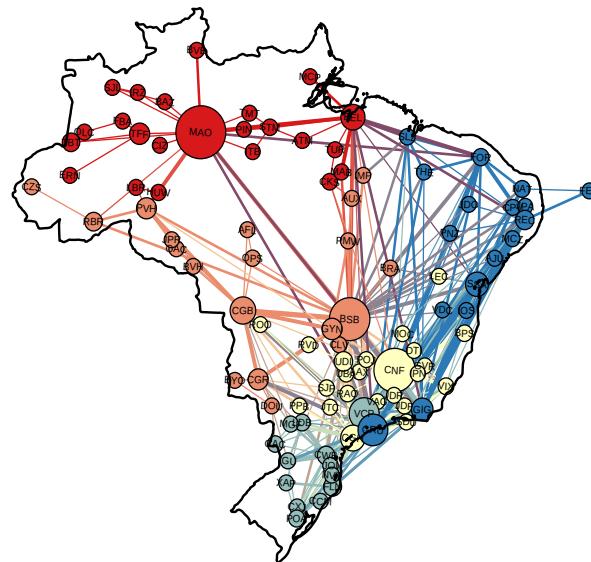
The data used in this paper was manually collected from websites of the airlines companies in the period between September and December 2012. The data available on the websites specifies if there is a regular route between two given airports, giving no details about the number of flights in a specific route or the number of passengers. We concentrated on the five largest airline companies that operate in Brazil (TAM, Gol/Webjet, Azul/Trip, Avianca and Passaredo), where largest is defined by the the number of passengers per kilometer flown. Together these five companies are responsible for more than 94% of the traffic of domestic and international flights. These flights take off and arrive in a total of 117 airports, out of which 22 are located outside Brazil and hence not used in the analysis. We have not collected the reminder 6% of flights because the smaller companies do not give us a way to extract the information from their websites, and the other international companies such as Delta and Lufthansa, would all have to be visited for us to collect just a handful of flights to the dataset.

In this work each airport is represented as a node in a network and there is an edge connecting these two nodes if there is a *direct flight* between the airports. We totalled 569 direct flights, out of which 31 are international. The network is weighted by the number of routes between the airports; e.g. if two airlines fly the route the weight of the connection is two. The network is also undirected due to the symmetry of direct flights—if there is a flight between airports A and B provided by company X then there is also a flight from airport B to A provided by the same company.

In our first analysis we decided to look at the betweenness of the nodes. Betweenness centrality is a measure of the extent to which a vertex is in the middle of the shortest path between two nodes [17]. In the case of airports we would like hubs to be the nodes with the highest betweenness as this may lead to shorter travel times. Figure 2 shows the betweenness of each node where nodes are sized based on their betweenness.

In the Figure 2 we can see that the nodes with highest betweenness are Manaus Airport (MAO), Brasilia Airport (BSB), Belo Horizonte Airport (CNF), Campinas Airport (VCP) and Guarulhos Airport (GRU). The airports MAO, CNF and VCP have high betweenness because they connect small airports inside of their own regions, instead of GRU and BSB that connect almost the entire country. Overall, GRU is not the airport with the highest betweenness although it is the largest airport in Brazil. One can also see that airports in the northeast part of the country have small betweenness which should translate in longer travel time to those locations. An improvement to this situation could be the creation of a local hub in the northeastern linking all the cities around the area and the major cities across the country.

We also performed a community detection in the network as proposed by Blondel et al. [18], which is a heuristic method based on modularity optimization

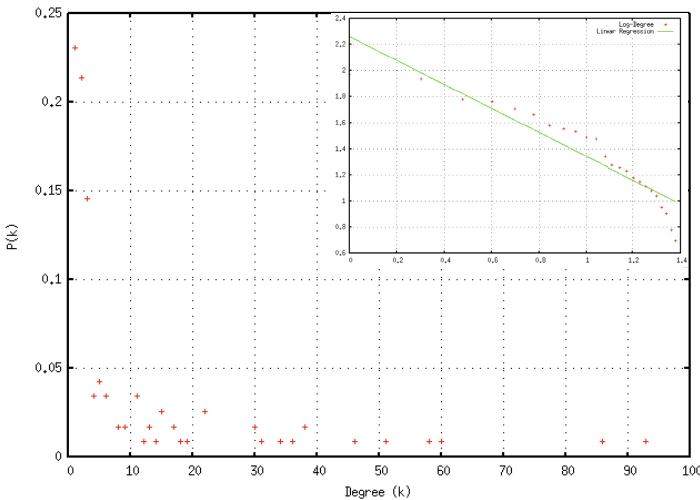


**Fig. 2.** Air transportation network for Brazil. The node size represent the betweenness of each airport (number of shortest paths going through that node). The colors represent the community analysis of the airports (airports that forms densely connected groups).

[19]. Each color in the Figure 2 represents a community. The north region of the country forms a community because of the large number of airports that are inter-connected. The same occurs in the central and south regions of the country. An interesting phenomenon occurs in the northeast region of the country, which has the second highest population in the country and the third highest GDP (Gross Domestic Product). Most of the airports in this region are in the same community of the hubs of the southeast region, Guarulhos Airport (GRU) and Rio de Janeiro Airport (GIG). This aspect indicates that the northeast airports are “dependent” of two aforementioned airports which may lead to delays in the entire northeast if a problem takes place in GRU or GIG.

In general terms the network has small-world characteristics. It has short average path length. For a network to be considered small world the average path length should be  $\simeq \log n$ , where  $n$  is the number of nodes of the network, and the network also needs to be clustered [20]. In the network of Brazilian flights the average path length is 2.68 and the average clustering coefficient is 0.64 (very high clustering). This characteristic is not surprising since the network is designed to provide a minimum number of hops between any two nodes.

The network also presents aspects of a scale-free network. To be a scale-free network the degree distribution of the nodes in the network must follow a power-law distribution with a negative exponent as in  $p(k) = k^{-\lambda}$ . This distribution represents the fact that very few nodes have high degree and the majority of nodes have a small degree (a long tail). Figure 3 clearly shows a long tail plus



**Fig. 3.** Degree distribution of airports in Brazil. The inset shows the distribution in a log-log plot.

just a few airport having high connectivity. To be considered a power-law distribution not only the long tail behavior is necessary, but also the exponent of the distribution should be  $2 \leq \lambda \leq 4$  [21]. When we do a fitting in the log-log plot (inset of Figure 3) we find that  $\lambda = 3.34$ . Hence we can say that the data follow a power-law distribution.

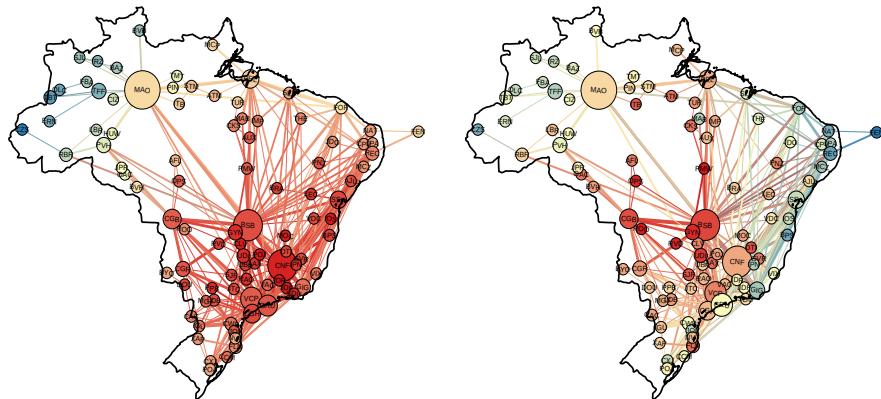
#### 4 Analysis of the Geolocation of Airport Hubs

The definition the best location to establish an airport hub depends on many factors including economic issues. However, setting the economic issues aside, it is possible to investigate for a particular country where the hub should be and the impact of that hub to the efficiency of the network.

First, we measured the average time of a flight that pass through the current main hub, Guarulhos Airport (GRU), in São Paulo, so we can compare it against the cases if others airports were the hub. For calculating the average time with GRU as the main hub, we chose an airport randomly and checked if it has a direct flight to GRU, if the link exist, we picked a random destination from GRU. One hundred flights with this configuration were chosen and for each one of them we estimated the flight time based on the direct distance between the airports. The average time of flights that have one connection in Guarulhos Airport is  $3:15 \pm 0:26$ hs for domestic flights while the average time for international flights that have one connection in Guarulhos Airport is  $20:03 \pm 0:39$ hs.

We estimated average flight time through the other airports in the same way we done it for GRU (above). The heat map constructed based on the average flights time can be seen in Figure 4. As we can see in the heat map of domestic

flights, Figure 4(left), the actual location of Guarulhos Airport (GRU) is not best average travel time, the best location tends to be where the density of airports in the country is higher, in the case of Brazil, nearer the center of the country where the airports of Belo Horizonte Airport (CNF) and Brasília (BSB) are located. Therefore in a single-hub model, and assuming the current configuration of airports in Brazil, the main hub should be around the two aforementioned airports.



**Fig. 4.** Heat map of average flight time of domestic (left) and international (right) flights. The closer to red is the color the best is the location as a hub.

In the heat map of international flights, Figure 4 (right), we can observe a behavior distinct from the heat map of domestic flights. This time one have the airports in the outer part of the country with worse average flight time than the central ones. The reason for the center to be the best location relates to the geographical location of Brazil. For flights in South America the eastern part of Brazil is in a disadvantage because the other countries in South America are located to the west and north of Brazil. For flights coming from Europe, the airports in the West of Brazil are in disadvantage and for flights coming from North America, the south of Brazil are in a disadvantage, the result is that Brasília (BSB) located in the center of the country appears with the best average time for international flights.

As we mention before, having only one hub may cause many problems, so in order to measure the impact of unavailability of a particular airport in Brazil, we calculated the number of direct flights, the number of flights with one stop and the number of flights with more than one stop in scenarios with the hub and without it. This approach also allow us to measure the importance of a given airport in the network.

In the actual scenario, with the hub at Guarulhos Airport (GRU), the number of possible direct flights is 346, the number of flights with one stop is 2,368 and the number of flights with more than one stop is 4,072. By removing this node of

the network we expect that the number of direct flights and the number of flights with one stop de-increase leading to an increase in the number of flights with more than one stop. So when we removed the Guarulhos airport of the network the number of direct flights decreased 17.6%, the number of flights with one stop decreased 37.5% leading to an increase of 23.3% in the number of flights with more than one stop.

We also estimated the impact of the removal of other nodes to measure comparatively its importance related to Guarulhos Airport. The removal of Fortaleza Airport (FOR) caused a decrease of 12.1% of direct flights, a decrease of 35.9% of flights with one stop and an increase of 22.5% of flights with more than one stop, which are better numbers than the removal of Guarulhos Airports. Similar results were found when we removed the Brasília Airport (BSB) and Manaus Airport (MAO), the hubs of the center and north regions respectively. Table 1 presents the number of flights under different scenarios.

**Table 1.** Number of flights per stop in scenarios that assume some airports do not exist

| Airport Removed | # Flights with more than one stop | # Flights with one stop | # Direct flights |
|-----------------|-----------------------------------|-------------------------|------------------|
| (none)          | 4,072                             | 2,368                   | 346              |
| GRU             | 5,022                             | 1,479                   | 285              |
| FOR             | 4,990                             | 1,492                   | 304              |
| BSB             | <b>5,179</b>                      | <b>1,326</b>            | <b>281</b>       |
| MAO             | 5,146                             | 1,343                   | 297              |

Table 1 demonstrates that the Brasília Airport (BSB) may lead to more disruption in the network than any other airports tested. Given that Brasília is the political capital of Brazil, there are many direct flights from and to it. Hence its removal lead to a smaller number of direct flights and a large increase of flights with two or more stops.

## 5 Conclusions and Future Work

In this work we evaluated some aspects of the Brazilian airline network. The network has small-world characteristics and the degree distribution follows a power law. We also identified communities in the network; these communities strongly correlate to the five regions in Brazil. However, some regions, such as the northeast, demonstrated high dependency of the main hubs of the network, not having a well defined community. We also calculated the betweenness centrality of each node in order to identify the most important nodes—those who the biggest number of shortest paths that pass through it. We found out that many nodes have high betweenness centrality because they are “local” hubs connecting a larger city to smaller airports in the region. We also found that the Brazil Brasília Airport (BSB) has a high betweenness because it has direct flights to most other major cities in Brazil.

In order to evaluate the dependency of the entire air transportation system of a particular airport we estimated the impact of removing strategic airports from the network by measuring the decrease in direct flights in the network and the increase in multiple-hop flights. The results shown that among the airports evaluated the BSB was the one that has the highest impact on the entire air transportation traffic if it becomes inoperative.

In this work we did not consider the number of daily flights between airports. We plan to collect the data related to the number of flights to perform a better analysis. Another way to make the evaluation more realistic is to include data about the amount of passengers who fly in each flight, because flights with more passengers contribute more to the air traffic. However, we are unaware of a way to get this data; we will work to estimate the number based on something proportional to the population of the location where the airport is located.

## References

1. Oliveira, A.V.M., Salgado, L.H.: Reforma regulatória e bem-estar no transporte aéreo brasileiro: e se a flexibilização dos anos 1990 não tivesse ocorrido? Documento de Trabalho N. 013—Acervo Científico do Núcleo de Estudos em Competição e Regulação do Transporte Aéreo, NECTAR (2006)
2. Lopes, F.S.: Estudo da evolução da estrutura de rotas das empresas aéreas no Brasil. Trabalho de Conclusão de Curso de Graduação—Instituto Tecnológico de Aeronáutica, São José dos Campos (2005)
3. Bergiante, N.C.R., Soares de Mello, J.C.C.B., Nunes, M.V.R., Paschoalino, F.F., et al.: Aplicação de uma proposta de medida de centralidade para avaliação de malha aérea de uma empresa do setor de transporte aéreo brasileiro. *Journal of Transport Literature* 5(4), 119–135 (2011)
4. Costa, T.F.G., Lohmann, G., Oliveira, A.V.M.: A model to identify airport hubs and their importance to tourism in Brazil. *Research in Transportation Economics* 26(1), 3–11 (2010)
5. Coelho, R.: O futuro da privatização no Brasil. *A Privatização no Brasil—O Caso dos Serviços de Utilidade Pública*, BNDES, Rio de Janeiro (2000)
6. Correia, T., Soares de Mello, J.: Avaliação da eficiência das companhias aéreas brasileiras com modelo dea nebuloso. In: *Transporte em Transformação XIII*, pp. 199–215. Gráfica Positiva, Brasília (2009)
7. Burghouwt, G., Hakfoort, J., van Eck, J.R.: The spatial configuration of airline networks in Europe. *Journal of Air Transport Management* 9(5), 309–323 (2003)
8. Guimera, R., Mossa, S., Turtschi, A., Amaral, L.A.N.: The worldwide air transportation network: Anomalous centrality, community structure, and cities' global roles. *Proceedings of the National Academy of Sciences* 102(22), 7794–7799 (2005)
9. Guimera, R., Amaral, L.A.N.: Modeling the world-wide airport network. *The European Physical Journal B—Condensed Matter and Complex Systems* 38(2), 381–385 (2004)
10. Pinheiro, M.C., Soares de Mello, J.: Ordenação dos aeroportos do Brasil através do apoio de análises multicritério. In: *Anais do XXXVII Simpósio Brasileiro de Pesquisa Operacional* (2005)
11. Pacheco, R.R., Fernandes, E.: Managerial efficiency of Brazilian airports. *Transportation Research Part A: Policy and Practice* 37(8), 667–680 (2003)

12. Guida, M., Maria, F.: Topology of the italian airport network: A scale-free small-world network with a fractal structure? *Chaos, Solitons & Fractals* 31(3), 527–536 (2007)
13. Bagler, G.: Analysis of the airport network of india as a complex weighted network. *Physica A: Statistical Mechanics and its Applications* 387(12), 2972–2980 (2008)
14. Zhang, J., Cao, X.-B., Du, W.-B., Cai, K.-Q.: Evolution of chinese airport network. *Physica A: Statistical Mechanics and its Applications* 389(18), 3922–3931 (2010)
15. Paleari, S., Redondi, R., Malighetti, P.: A comparative study of airport connectivity in china, europe and us: which network provides the best service to passengers? *Transportation Research Part E: Logistics and Transportation Review* 46(2), 198–210 (2010)
16. Endres, G.: Crisis control: troubled year highlights problems in brazils air transportation market. *Flight International* (2007)
17. Newman, M.E.J.: A measure of betweenness centrality based on random walks. *Social Networks* 27(1), 39–54 (2005)
18. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 2008(10), P10008 (2008)
19. Newman, M.E.J.: Modularity and community structure in networks. *PNAS* 103(23), 8577–8582 (2006)
20. Watts, D., Strogatz, S.: The small world problem. *Collective Dynamics of Small-World Networks* 393, 440–442 (1998)
21. Albert, R., Barabási, A.L.: Statistical mechanics of complex networks. *Reviews of Modern Physics* 74(1), 47 (2002)

# Author Index

- Ahmed, Syed Ali 421  
Albers, Susanne 1  
Angel, Eric 316  
Angelucci, Anna 17  
Asgari, Omid 841  
Ashok, Pradeesha 221
- Babu, Jasine 626  
Bampis, Evripidis 134  
Banik, Aritra 197  
Baron, Joshua 169  
Basavaraju, Manu 626  
Baumbach, Jan 373  
Bazgan, Cristina 543  
Ben-Ameur, Walid 337  
Bi, Yuanjun 737, 780, 821  
Bianchi, Maria Paola 53  
Bilò, Davide 5  
Bilò, Vittorio 5, 17  
Böckenhauer, Hans-Joachim 53, 493  
Braverman, Vladimir 638
- Campêlo, Manoel B. 614  
Cao, Zhigang 122  
Carvalho, Marco 849  
Cha, Jianzhong 603  
Chandran Leela, Sunil 626  
Chen, Danny Z. 603  
Chen, Jianer 89, 555, 567  
Chen, Kun-Tze 409  
Chen, Xujin 122  
Cheng, Jingde 697  
Cheong, Otfried 77  
Chin, Francis Y.L. 506  
Chiplunkar, Ashish 481  
Chlebík, Miroslav 280  
Chlebíková, Janka 280  
Chopin, Morgan 543  
Crespelle, Christophe 469  
Crowston, Robert 434  
Cui, Wenjuan 385
- Damaschke, Peter 446  
Das, Sandip 197  
da Silva, Maise Dantas 531
- Du, Donglei 292, 304  
Duan, Zhenhua 158, 591
- Elbassioni, Khaled 65  
Elmasry, Amr 147  
El Shawi, Radwa 77
- Fan, Lidan 737, 780  
Fekete, Sándor P. 208  
Feng, Qilong 89, 567  
Flammini, Michele 17  
Fouque, Pierre-Alain 651  
Friedrichs, Stephan 208  
Fu, Bin 713  
Fujiwara, Hiroshi 518
- Gao, Hongbiao 697  
Gelles, Ran 638  
Georgiou, Konstantinos 29  
Goebel, Randy 257  
Goto, Yuichi 697  
Govindarajan, Sathish 221  
Greenbaum, Nancy L. 421  
Grindrod, Peter 791  
Gudmundsson, Joachim 77  
Guo, Jiong 373, 555  
Guo, Longkun 325  
Gutin, Gregory 434
- Han, Meng 747  
Han, Xin 506  
He, Jia 591  
Hell, Pavol 579  
Hossain, Md. Iqbal 672  
Hromkovič, Juraj 53, 493  
Hu, Tao 663  
Huiban, Cristiana G. 614
- Ibragimov, Rashid 373  
Ishai, Yuval 169  
Ito, Takehiro 729
- Ji, Shouling 747  
Jiang, Haitao 397  
Jones, Mark 434  
Juhl, Daniel Dahl 147

- Karakostas, George 29  
 Katajainen, Jyrki 147  
 Kern, Walter 41  
 Khopkar, Abhijit 221  
 Kim, Donghyun 841  
 Kisielewicz, Andrzej 182  
 Kleinberg, Robert 4  
 Kobayashi, Koji 518  
 Kobayashi, Yasuaki 458  
 Komm, Dennis 493  
 Könemann, Jochen 29  
 Kononov, Alexander 134  
 Kowalski, Jakub 182  
 Kröller, Alexander 208  
 Krug, Sacha 53, 493  
 Kumar, Rahul 831
- Lambert, Thomas 469  
 Letsios, Dimitrios 134  
 Li, Chih-Hsuan 268  
 Li, Chi-Long 409  
 Li, Deying 841  
 Li, Jinbao 747  
 Li, Minming 101  
 Li, Shaohua 89  
 Li, Weidong 385  
 Li, Wenfeng 713  
 Li, Yingshu 747, 841  
 Li, You 801  
 Li, Yu 292  
 Li, Yue 770  
 Lian, Biao 821  
 Liang, Qilian 801  
 Liao, Kewen 325  
 Lin, Guohui 257  
 Lin, Yu-An 361  
 Liu, Benyuan 801  
 Liu, Nan 397  
 Liu, Peihai 114  
 Liu, Tao 759  
 Liu, Tian 721  
 Liu, Xingwu 705  
 Liu, Yunlong 555  
 Lu, Chin Lung 409  
 Lu, Junzuo 770  
 Lu, Xiwen 114  
 Lu, Yiping 603  
 Lu, Zaixin 737  
 Lu, Zhao 721  
 Lucarelli, Giorgio 134
- Maheshwari, Anil 197  
 Makino, Kazuhisa 65  
 Mans, Bernard 349  
 Maruta, Hirokazu 458  
 Mathieson, Luke 349  
 Mehlhorn, Kurt 65  
 Menezes, Ronaldo 849  
 Mishra, Aurosish 579  
 Misra, Neeldhara 221  
 Mneimneh, Saad 421  
 Mohajer, Meysam 811  
 Mohamed-Sidi, Mohamed-Ahmed 337  
 Morishita, Shiho 245  
 Moscardelli, Luca 17  
 Mu, Zongxu 101  
 Muciaccia, Gabriele 434  
 Muhammad, Azam Sheikh 446
- Nakae, Yusuke 458  
 Nemparis, Ioannis 134  
 Neto, José 337  
 Nichterlein, André 543  
 Nip, Kameng 680  
 Nishizeki, Takao 245
- Okada, Taku 729  
 Oliveira, Douglas 849  
 Ostrovsky, Rafail 169, 638
- Peng, Chao 688  
 Peng, Zhiyong 713  
 Poon, Chung Keung 506  
 Poon, Sheung-Hung 361  
 Protti, Fábio 531
- Qiang, Yan 759, 770  
 Qiu, Xian 41
- Rahman, Md. Saidur 672  
 Rajendraprasad, Deepak 626  
 Rajgopal, Ninad 221  
 Ramezani, Fahimeh 65  
 Rastegarnia, Adib 811  
 Regnault, Damien 316  
 Ren, Xianyi 663
- Sampaio, Rudini M. 614  
 Satti, Srinivasa Rao 147  
 Schmidt, Christiane 208  
 Shen, Hong 325  
 Sherette, Jessica 233  
 Shi, Feng 567

- Shi, Kai 697  
Sikora, Florian 543  
Singh, Anurag 831  
Singh, Yatindra Nath 831  
Smid, Michiel 197  
Smula, Jasmin 493  
Solouk, Vahid 811  
Souza, Uéverton dos Santos 531  
Sprock, Andreas 493  
Stamirowska, Zuzanna 29  
Steffen, Björn 53  
Stoyanov, Zhivko 791  
Suzuki, Akira 729  
Szykuła, Marek 182
- Tamaki, Hisao 458  
Teng, Shang-Hua 705  
Thang, Nguyen Kim 316  
Thierry, Eric 469  
Tian, Cong 158, 591  
Ting, Hing-Fung 506  
Tokuta, Alade O. 841  
Tong, Weitian 257  
Tsin, Yung H. 506
- Vergnaud, Damien 651  
Vishwanathan, Sundar 481  
Vukadinović Greetham, Danica 791
- Wakabayashi, Yoshiko 614  
Wang, Ailian 737, 780  
Wang, Biing-Feng 268  
Wang, Changjun 122
- Wang, Jianxin 89, 555, 567  
Wang, Jie 801  
Wang, Li 821  
Wang, Lusheng 385  
Wang, Zhenbo 680  
Wu, Chenchen 304  
Wu, Lidong 759, 770  
Wu, Weili 737, 759, 770, 780, 821
- Xiu, Naihua 292  
Xu, Chao 555  
Xu, Dachuan 292, 304  
Xu, Ke 721  
Xu, Wen 821
- Yan, Mingyuan 747  
Yang, Chung-Han 409  
Yang, Mengfei 158, 591  
Ye, Deshi 506  
Ye, Jhih-Hong 268  
Yoon, Sang Duk 233
- Zapalowicz, Jean-Christophe 651  
Zhang, Jihong 663  
Zhang, Xiaolong 759, 770  
Zhang, Yong 506  
Zhao, Juanjuan 759, 770  
Zhou, Jie 688  
Zhou, Xiao 729  
Zhu, Binhai 397, 688  
Zhu, Daming 397  
Zhu, Hong 688